THESIS

MOBILE COMPUTING SOLUTIONS FOR EFFECTIVE AND EFFICIENT GENERATION AND DISSEMINATION OF TACTICAL OPERATIONS ORDERS

by

Paul D. Haagenson

June 2017

Thesis Advisor:	John Gibson
Co-Advisor:	Gurminder Singh

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2017	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE MOBILE COMPUTING SOLUTIONS FOR EFFECTIVE AND EFFICIENT GENERATION AND DISSEMINATION OF TACTICAL OPERATIONS ORDERS			**5. FUNDING NUMBERS**
6. AUTHOR(S) Paul D. Haagenson			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Leaders of all grades and occupational specialties in the Marine Corps and throughout the military issue operations orders to communicate intended action in tactical environments. Despite advances in communication and mobile computing technology, most leaders in austere environments still create orders with pen and paper and disseminate them verbally or with manual copying. This process is inefficient and error-prone. We propose an Android application, the Tactical Operations Order Tool–Handheld (TOOTH), to increase the effective development and dissemination of tactical operations orders by expeditionary leaders. Our research utilizes the author's experience as an instructor and practitioner in conjunction with Marine Corps and Joint doctrine, to establish a new paradigm for capturing, storing and transmitting the informational elements of an operations order. We developed a working technology demonstrator that incorporates a new object-oriented data structure into existing open-source mobile technology to provide a framework for future development and testing. We received positive feedback on the initial application design, and early results indicate time savings over the manual process in excess of 50%. We believe that a fully implemented version of this capability has the potential to increase mission planning effectiveness, decrease errors, and save tens of thousands of person-hours annually.

14. SUBJECT TERMS operations, orders, mobile, application, Android, five paragraph, mobile, map, software			**15. NUMBER OF PAGES** 141
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

THIS PAGE INTENTIONALLY LEFT BLANK

MOBILE COMPUTING SOLUTIONS FOR EFFECTIVE AND EFFICIENT GENERATION AND DISSEMINATION OF TACTICAL OPERATIONS ORDERS

Paul D. Haagenson
Major, United States Marine Corps
B.S., United States Naval Academy, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2017

Approved by: John Gibson
Thesis Advisor

Dr. Gurminder Singh
Thesis Co-Advisor

Dr. Peter Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Leaders of all grades and occupational specialties in the Marine Corps and throughout the military issue operations orders to communicate intended action in tactical environments. Despite advances in communication and mobile computing technology, most leaders in austere environments still create orders with pen and paper and disseminate them verbally or with manual copying. This process is inefficient and error-prone. We propose an Android application, the Tactical Operations Order Tool–Handheld (TOOTH), to increase the effective development and dissemination of tactical operations orders by expeditionary leaders. Our research utilizes the author’s experience as an instructor and practitioner in conjunction with Marine Corps and Joint doctrine, to establish a new paradigm for capturing, storing and transmitting the informational elements of an operations order. We developed a working technology demonstrator that incorporates a new object-oriented data structure into existing open-source mobile technology to provide a framework for future development and testing. We received positive feedback on the initial application design, and early results indicate time savings over the manual process in excess of 50%. We believe that a fully implemented version of this capability has the potential to increase mission planning effectiveness, decrease errors, and save tens of thousands of person-hours annually.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AI	Artificial Intelligence
API	Application Program Interface
BML	Battle Management Language
C2PC	Command and Control Personal Computer
C-BML	Coalition Battle Management Language
COA	Course of Action
CPOF	Command Post of the Future
DAG	Directed Acyclic Graph
DOD	Department of Defense
GPS	Global Positioning System
GRG	Grid Reference Graphic
HTML	Hyper-text Markup Language
IDE	Integrated Development Environment
IEEE	Institute of Electronics and Electrical Engineers
J-BML	Joint Battle Management Language
JMPS-E	Joint Military Planning System – Expeditionary
MaS	Modelling and Simulation
MAGTF	Marine Air-Ground Task Force
MCPP	Marine Corps Planning Process
MTWS	MAGTF Tactical Warfare Simulator
NATO	North Atlantic Treaty Organization
NPS	Naval Postgraduate School
OTIC	Operations and Tactics Instructor Course
SOM	Scheme of Maneuver
TBS	The Basic School
TCM	Tactical Control Measure
TOOTH	Tactical Operations Order Tool - Handheld
U.S.	United States
USMC	United States Marine Corps
WBS	Work Breakdown Structure

WWI	World War One
WWII	World War Two
XML	Extensible Markup Language

ACKNOWLEDGMENTS

I would like to thank my advisors for taking my ideas and shaping them into a worthwhile project. Their guidance and expertise has been invaluable.

I would like to thank my lovely wife for being my best friend and for her constant support in allowing me to do the job that I love, even when it has resulted in deployments and more work for her. I would like to thank my adorable children for giving me extra reasons to enjoy coming home every day.

I would like to thank my peers for helping to smooth my transition into the world of computer science.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Less than two years from today, a United States Marine Corps (USMC) lieutenant will be sitting against the wall in a dark room in a nameless building in a nameless country in the Middle East. He will be subconsciously listening to his Marines as they finish clearing the upper stories of the building and establish perimeter security, but his focus is on the task that he just received over the radio: a change of plans. His company is no longer going to be clearing their current sector of the city but has instead been tasked to conduct a movement-to-contact several kilometers into an assault on a suspected enemy strong point along the nearby river. His platoon will be the lead element and he has been given a series of checkpoints to follow, along with the last known information about the enemy's positions. The company moves in four hours, but the company commander wants to be back-briefed on the Lieutenant's plan in one hour. His platoon sergeant enters the room and says that he heard the word has changed; he wants to know what he can tell the squad leaders. "Tell them we're walking," says the Lieutenant. "Tell them to change their socks and to spread-load First Squad's gear. I'll tell them more once I have a plan." He rubs his forehead, tired like the others after seven straight days of nerve-racking and bone-numbing building clearing. He slaps himself in the face and forces himself to focus.

He knows how this used to be done, the way he was taught at the Infantry Officer Course (IOC). He remembers the stress of frantically scribbling words on write-in-the-rain paper with a grease pencil, of making side-lists of any items to revisit in order to keep his entire plan synchronized. He still carries a paper map, compass, GPS receiver, and combat orders template in his pack just in case. But he doesn't need it today. He reaches instead into his cargo pocket and pulls out the Marine Corps-issued Android tablet that he has been using to track friendly and enemy positions all day. Still mostly charged from the night before, it is now time to put it to work in a different way. He opens the tactical planning application, and is greeted with a series of simple questions and easy-to-press buttons. He selects the option to build a new operations order and selects the type of mission from a list of templates. He loads information previously saved about his company's units and their locations with a few more taps on the tablet's screen. He flips over to the map that is built into the application and quickly adjusts the friendly and enemy locations based on the information he was given on the radio. He changes the map from satellite imagery to a topographic layer and, with a few more clicks, he adds the checkpoints he has been told to use. He stares at the map for a few minutes, seeing a plan develop in his mind.

He knows what he wants to do. He also recognizes his own mental exhaustion, so he taps the button that says, "Guide Me." The application asks if he wants to load the standard tasks for a movement-to-contact, and then walks him through the sequencing of those tasks, working backwards from the objective. As he makes adjustments to the tasks, they appear on a continuously-updating timeline and he is able to see that his plan is tactically sound. He adds an engineering unit that he forgot was attached to his platoon. All of his adjustments have been automatically plotted on the map, with graphics depicting the tactical tasks to be accomplished. He clicks the "Check Me" button, and the application reminds him to assign a task to the engineers. It also points out some potentially dangerous problems with his geometry of fires. Several quick fixes and the application is "happy." The plan is sound.

He calls for his squad leaders and uses a small hand-held projector to beam the plan onto the grimy walls of the building. He flips to another screen that lists the tasks for each squad and a set of recommended gear checks and inspections that are customized to those tasks. As the squad leaders move out to prepare their Marines, the Lieutenant's fellow platoon commander comes through the door after running from the next sector over. He pulls out his own tablet and in a minute the operations order has been wirelessly transferred. They talk about a few details, shake hands, and part ways. Within a few minutes that other leader will sit in his own dark room. A few taps on his tablet, and he is presented with the option to make a new order for his own platoon based on the existing one. The company commander is briefed with fifteen minutes to spare.[1]

A. TECHNOLOGY GAP

The Five Paragraph Order Format for operations orders has existed in its current form since 1954, and is universally understood as a means of communicating intended military actions to all levels of the chain of command. Marine Corps officers are steeply versed in the format during their basic officer training, but the highly administrative process can distract from a focus on communicating sound tactical thought. Tactical leaders in the Marine Corps working in austere environments currently and primarily rely on manual methods for generating and disseminating operations orders.

Currently, both the hardware and software capabilities exist to develop a tool that would help tactical-level leaders build and disseminate orders more efficiently and accurately, but those technologies have not been combined at the mobile device level. A shift in the paradigm of the way we view the operations order information allows for the application of object-oriented programming design to the problem of creating and disseminating sound plans. This research examines those information elements and attempts to find the best approach to addressing the needs of the warfighter. This research will produce a working but tightly scoped technology demonstrator that will serve as a framework for future research and development efforts.

B. SYSTEM PURPOSE

The purpose of the proposed system, which the authors have named the Tactical Operations Order Tool–Handheld (TOOTH), is to make the preceding scenario a reality by treating the information in an operations order as a set of interconnected objects replacing the current highly-linear, paper-based format. If these information objects can be defined and linked, then those linked objects

[1] This scenario is based on the primary author’s personal experience as a platoon commander in Iraq in 2005, except for the existence of mobile technology.

can be saved as a completed plan. If the resulting data structure can be represented in an easy-to-use application, then the orders-writing process becomes accessible to a greater number of Marines, even in austere environments. A user only needs to define each object and relationship once, at which point the completed set of objects can be exported in any number of formats, including traditional written formats or non-traditional formats such as simulator definition files or detailed fire support documents. Those same linkages between objects can be examined in order to check the completed order for irregularities or potentially dangerous omissions. This capability has the potential to make leaders at all levels more efficient and accurate in generating and disseminating operations orders. Moreover, the system could be used as a training tool to improve the tactical thought of less experienced leaders.

C. RESEARCH QUESTIONS

This thesis strives to answer the following question:

- How can we automate the best practices of operations order generation in order to improve effectiveness and efficiency?

The following sub-questions parse the primary question:

- Which artificial intelligence tools and data structure algorithms are best suited to aiding the manual creation of operations orders while simultaneously identifying and resolving discrepancies, and how might they be utilized for the desired capability?
- Which data structures most accurately capture the information contained in an operations order while still allowing portability and flexibility in a digital format, and how might they be integrated to form an intuitive application?
- Which principles of human/computer interface design can be employed to increase efficiency and accuracy of orders generation?
- Can the tools identified to increase orders generation efficiency be employed within the framework of existing digital interoperability products?

- Does new software created using these tools show potential to improve the ability of USMC leaders to quickly and accurately generate operations orders?

D. SCOPE AND BOUNDARIES

This thesis focuses on making the tactical operations order process more accurate and efficient through the use of mobile computing technology. It envisions a mobile application that combines an object-oriented data structure and a mapping component with the ability to template information. It shapes the data for import and export in a variety of formats. This thesis does not provide a fully-functional system due to the limitations of time and development resources; rather, it will create a partially developed application to serve as a technology demonstrator and a framework upon which to add future capability. The goal hierarchy discussed in Chapter III and included in Appendix A serves as a vision of what the authors believe is ultimately possible—several major features will not be fully implemented in the technology demonstrator. While every effort has been made to exercise sound practices with respect to application design and coding, any formal continuation of this work would benefit from review by professional programmers.

TOOTH is designed for use by Marines who have completed their entry level training and Military Occupational Specialty (MOS) schooling, and have therefore been exposed to the basics of operations orders. The technology assumes some familiarity with the process of writing and disseminating those orders and also assumes that the user meets health requirements for continued military service. The application is designed with the needs of the military user in mind and does not provide accommodation to persons with vision or other disabilities.

The technology demonstrator is designed to be stand-alone. Initial efforts to integrate with existing technologies were abandoned in the interest of reducing complexity and dependency on external support during the development process; however, the authors recognize the value of having the technology interface with existing programs and software systems, and encourage focus on interoperability in future work. Additionally, the technology demonstrator is built to run on specific Android devices. A full product should support scaling for a wide range of device types and capabilities, but achieving that level of accessibility was not necessary to demonstrate the core ideas of this research.

E. RELEVANCE TO THE DEPARTMENT OF DEFENSE

The Department of Defense (DOD) possesses operational planning systems, mapping technologies, mobile device technologies, and a series of formats and language constructs for capturing the information contained in tactical operations orders. It currently has no system that combines all of these features into a single tool for leaders at the tactical level. The creation and adoption of a mobile computing solution for generating and disseminating operations orders at the tactical level has the potential to save tens of thousands of person-hours per year while simultaneously improving the accuracy and lethality of leaders operating in austere environments.

F. INITIAL RESULTS

Although TOOTH has not been formally tested in any way, early results suggest that its usage will result in efficiency gains well in excess of the 10% development target. For example, the amount of time needed for TOOTH to convert an existing operations order into a shell for a new order, with relevant original information moved to the appropriate locations in the new shell, is less than ten seconds. This represents a time-savings of more than 90% over similar actions in a manual process.

G. THESIS ORGANIZATION

Chapter I introduces this research by outlining the technology gap in the current tactical operations order process, introduces TOOTH as a potential solution, discusses the relevance of this research to the DOD, specifies the research questions to be addressed, and describes the scope and limitations of this thesis.

Chapter II discusses the background of the operations order, the current methods for creating and disseminating operations orders (along with their pitfalls), the existing technology, and a possible merger of ideas to create a solution to the current technology gap.

Chapter III discusses the design of the application. The requirements and goal structure for the application are detailed, to include the constraints imposed by working with existing Marine Corps mobile devices. We then detail the overarching design of the application, to include interfaces between the user, the software, and existing hardware and operating system features. The concept of object-oriented programming is reviewed, and the proposed data structures are outlined along with descriptions of how they will meet the requirements of the application. Finally, the proposed user interface is explained in the context of predicted user requirements and environmental limitations.

Chapter IV discusses the implementation of the design detailed in Chapter III and describes the creation of the technology demonstrator, including the choice of software development methodology, the design of the application data structure, and the algorithms required to implement the data structure effectively. This chapter also details the user interface implementation and the methods used to display order information within the interface. It concludes with a discussion of software testing and the results of initial demonstrations of the software to community representatives.

Chapter V concludes the thesis by discussing the effectiveness of the technology demonstrator's data structure and user interface. The chapter also details recommendations for both the adoption of the ideas presented and for the further development of TOOTH. It outlines potential future research projects in expert system theory, artificial intelligence, natural language processing, and military simulations, all of which have direct correlation to objectives listed in the software goal hierarchy.

II. BACKGROUND

A. THE OPERATIONS ORDER

For as long as there have been conflicts between organized militaries, there has been a requirement for military leaders to effectively communicate their intent to their subordinates. The form taken by those communications, or orders, has changed over time, but the overall purpose has not—to communicate a plan and to coordinate the actions of various units across time, space, and contingencies. The United States (U.S.) military and its allies are no different. Many leaders have debated and worked to find the most efficient way for commanders to communicate their plans and intent to subordinates, and after every major conflict in our nation's history there always follows much soul searching and deliberation about the standard processes (Smith 1989). This chapter examines the benefits and challenges of using the current standard tactical-operations order format and focuses specifically on its use within the United States Marine Corps (USMC), but the principles discussed can be applied to almost any format.

1. History

In his seminal book *Command in War*, Martin Van Creveld (Van Creveld 1985) identifies six unique systems of command and control of military operations throughout history, ranging from leadership on the literal front lines during melee combat to today's heavily technology-dependent and geographically-dispersed leadership challenges. He particularly highlights two driving factors that led to our current paradigm of command and control—the switch from oral to written orders between 1750 and 1850 (and the resulting relentless pursuit of the perfect format), and the advent of technology allowing for instantaneous communications across great distances (radio and satellite voice communications) (Van Creveld 1985).

In a 1989 thesis for the School of Advanced Military Studies, Major Matthew Smith explains that the five-paragraph format currently used by the United States and its allies has its roots in work introduced by a cavalry officer, Eben Swift, in 1897.

> In general terms, Swift's orders were clear, short, precise, and complete. They avoided every form of expression that could have been misunderstood because experience showed that such orders had invariably been misunderstood. Swift's order format used positive terms so that responsibility could be placed with ease. Orders were complete in form and legible even by a bad light. They avoided conjectures, expectations, reasons or apologies for measures taken. No order was given for things which would ordinarily be done without special instructions. (Smith 1989, 5)

The format was modified again after both WWI and WWII, and finally took its more modern form in 1954. The details of the schemes of maneuver were moved deeper into the format, giving recipients context for their tasks before telling them what to do. Compared to previous formats, this modern version is "more detailed and highly structured, having a paragraph for almost each type of information" (Filiberti 1987, 29).

2. Usage

The U.S. services and our North Atlantic Treaty Organization (NATO) allies all use the same basic structure of the tactical operations order, with only minor variations (U.S. Joint Forces Command 2011; NATO Military Agency for Standardization 2000). This leads to an ease of communication and interoperability that otherwise might not exist, especially where differences in language or operational experience create predictable difficulties. The format is used extensively within the U.S. military, not only by the Army and Marine Corps but also by ground combat elements in other services such as the Navy civil engineering units and Air Force combat air controllers.

Within the Marine Corps, the Five Paragraph Order format is used by all ground Military Occupational Specialties (MOSs), but is most pervasive in combat units and those who directly support combat units. All Marine Corps Officers undergo the same leadership training at The Basic School (TBS), where they are first taught to be provisional rifle platoon commanders before learning their specific MOS skills. The Five Paragraph Order format is used for communication of tactical ideas during all tactical field events and tactical decision games at the school, and thus its structure provides the framework for a large portion of basic Marine Corps officer training. Five horizontal themes of officership guide all training at The Basic School; one particularly pertinent to this study is to be "able to decide, communicate, and act in the fog of war" (The Basic School 2015b, 11). The Five Paragraph Order is expressly listed as the vehicle for training this ability (The Basic School 2015b). Due to this shared training background, the format is more engrained in the Marine Corps ethos and shared tradition than with other services. It is arguably the one commonly shared language of the entire Marine Corps officer community.

In order to understand the primary assertion of this thesis, it is first helpful to understand how much time is spent creating and disseminating operations orders every year. There is a surprising lack of research and data available, but we can derive an estimate from the author's experiences serving in several different infantry battalions and from conversational anecdotes with peers from other services. There are 24 active duty infantry battalions in the Marine Corps (United States Marine Corps 2017), each of which we estimate contains approximately 135 leaders who develop or brief tactical operations orders as part of their duties. Although actual numbers vary by specific billet, we estimate that each of these leaders averages six hours per month generating, reading, or disseminating tactical orders. As shown in Equation (1), this estimate predicts the total number of hours devoted to orders development in the Marine Corps infantry community is in excess of 233,000 hours per year.

$$24 \text{ battalions x } \frac{135 \text{ leaders}}{\text{battalion}} = 3,240 \text{ leaders} \tag{1}$$

$$3{,}240 \text{ leaders x } \frac{6 \text{ hours}}{\text{month}} \text{ x } \frac{12 \text{ months}}{\text{year}} = \frac{233{,}280 \text{ leader hours}}{\text{year}}$$

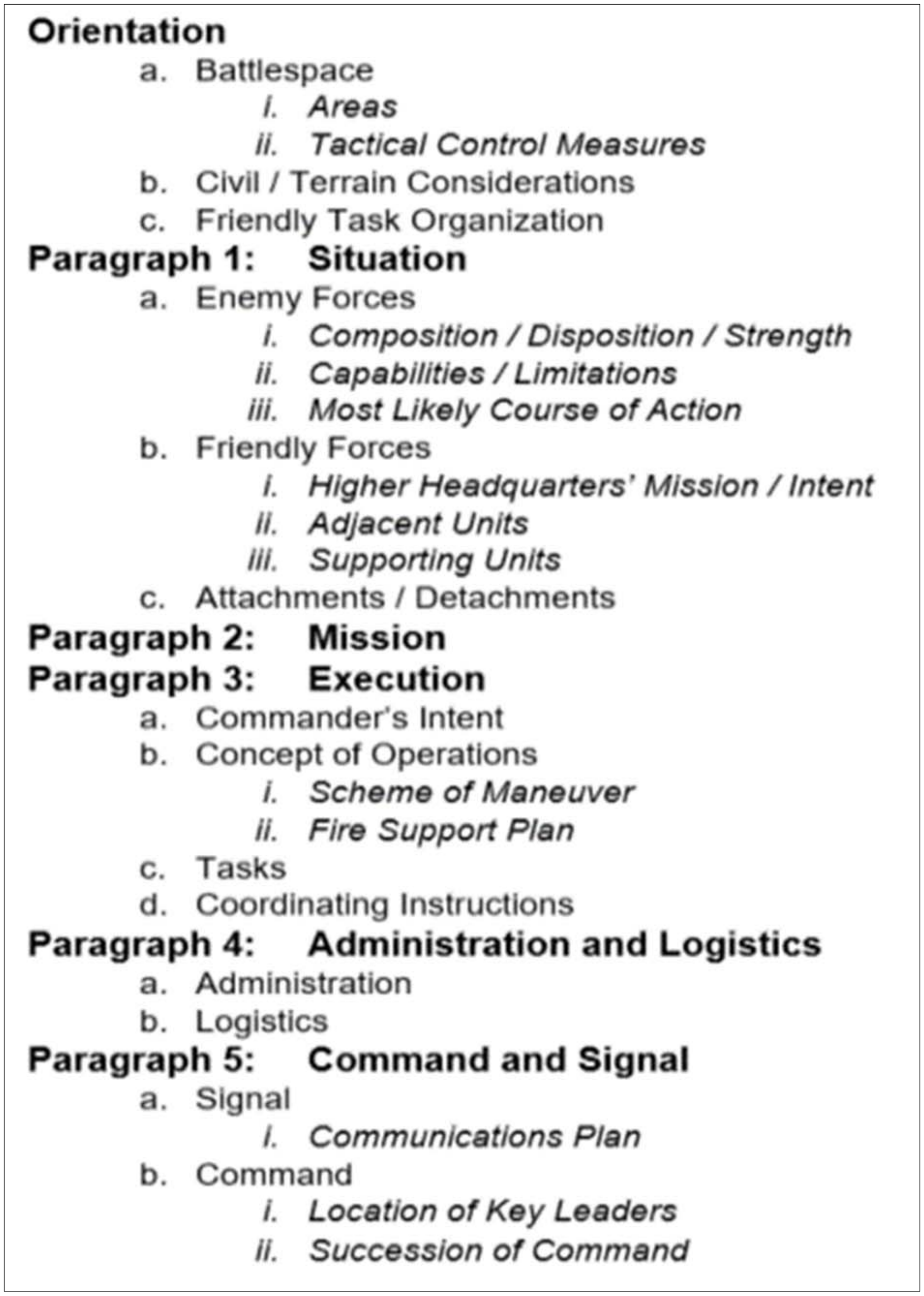
Even small increases in efficiency with respect to the way that the Marine Corps writes and disseminates tactical orders have huge potential benefits in terms of saving of time and labor. In the Marine Corps infantry alone, a ten percent improvement in the process would save more than 23,300 hours of leaders' time per year. This is approximately 10.8 person-years or full-time equivalent (FTE). Given that the annual DOD composite rate for a junior grade Marine Corps officer (O-3) in 2017 is approximately $139,000, this represents a minimum annual savings of $1,501,200 (Comptroller 2016). Expanding this result to include training commands and other MOSs such as Artillery, Combat Engineering or Logistics would realistically cause this number to double. Given the relative sizes of the U.S. Army and Marine Corps, the calculated total would triple if expanded to include the active-duty Army, and would expand further if other services and allied nations who use the same orders format are included. Additional potential savings exist in the form of efficiencies in training with regard to development of mission orders and reductions in human-induced errors in orders development.

3. Format

In order to understand where anticipated efficiencies of 10 or more percent can be found in the writing and disseminating operations orders, it is necessary to understand the structure and complexities of the format. Although the additions and other changes to the order that we have previously examined have been made for good reason, it is also not hard to understand why new Marine Corps officers spend the majority of their six month basic officer training immersed in it, and why so many of them end up falsely equating a well-formatted order with a solid plan.

a. Basic Structure

The tactical operations order format is comprised of five main paragraphs, each with various subparagraphs and associated lists and graphics. That structure is depicted in Figure 1, and is formulated around Swift's idea that the recipient of the order needs to understand context (Paragraph 1) in order to conceptualize their tasks. The primary task of the unit (Paragraph 2 – Mission) takes priority over all other information in the order, which is why it is presented as soon as the recipient can process it in the proper context. When a unit understands their primary task and purpose, everything else becomes additional amplification and clarification (Filiberti 1987). Paragraph 3 gives an overview of the plan, conveys specific tasks to subordinate units, and details the use of supporting arms in achieving those objectives. Paragraph 4 provides detailed administrative and logistics coordination, and Paragraph 5 provides information about how the operation will be commanded and controlled.



Orientation

- a. Battlespace
  - *i. Areas*
  - *ii. Tactical Control Measures*
- b. Civil / Terrain Considerations
- c. Friendly Task Organization

Paragraph 1: Situation

- a. Enemy Forces
  - *i. Composition / Disposition / Strength*
  - *ii. Capabilities / Limitations*
  - *iii. Most Likely Course of Action*
- b. Friendly Forces
  - *i. Higher Headquarters' Mission / Intent*
  - *ii. Adjacent Units*
  - *iii. Supporting Units*
- c. Attachments / Detachments

Paragraph 2: Mission

Paragraph 3: Execution

- a. Commander's Intent
- b. Concept of Operations
  - *i. Scheme of Maneuver*
  - *ii. Fire Support Plan*
- c. Tasks
- d. Coordinating Instructions

Paragraph 4: Administration and Logistics

- a. Administration
- b. Logistics

Paragraph 5: Command and Signal

- a. Signal
  - *i. Communications Plan*
- b. Command
  - *i. Location of Key Leaders*
  - *ii. Succession of Command*

Figure 1. Basic Five Paragraph Order Structure. Adapted from The Basic School (2015a).

### *b.	Internal Dependencies*

The overall operations order format appears simple, but it hides the fact that much of the information it contains is inter-connected. For example, information about tactical control measures (TCMs) such as unit objectives or phase lines that is given in Paragraph 1 of the order is linked to the specific actions that Marines will take at each of those TCMs. The type of operation that the unit is conducting shapes the unit's overarching task and implies that subordinate units will be assigned certain individual tasks. The operation type may also imply that the order will contain specific communications protocols or require specific rehearsals to be conducted prior to execution. The amount of communications equipment needed, its distribution among the unit, and the appropriate signals that are listed in Paragraph 5 all have their roots in the scheme of maneuver, fire support plan, and tasks listed in Paragraph 3. These are just a few examples from a list of interdependencies within the order that could be overwhelmingly large depending on the myriad variables associated with the entire spectrum of military operations. An example depicting the relationship of tactical tasks to other information within the order is given in Figure 2.

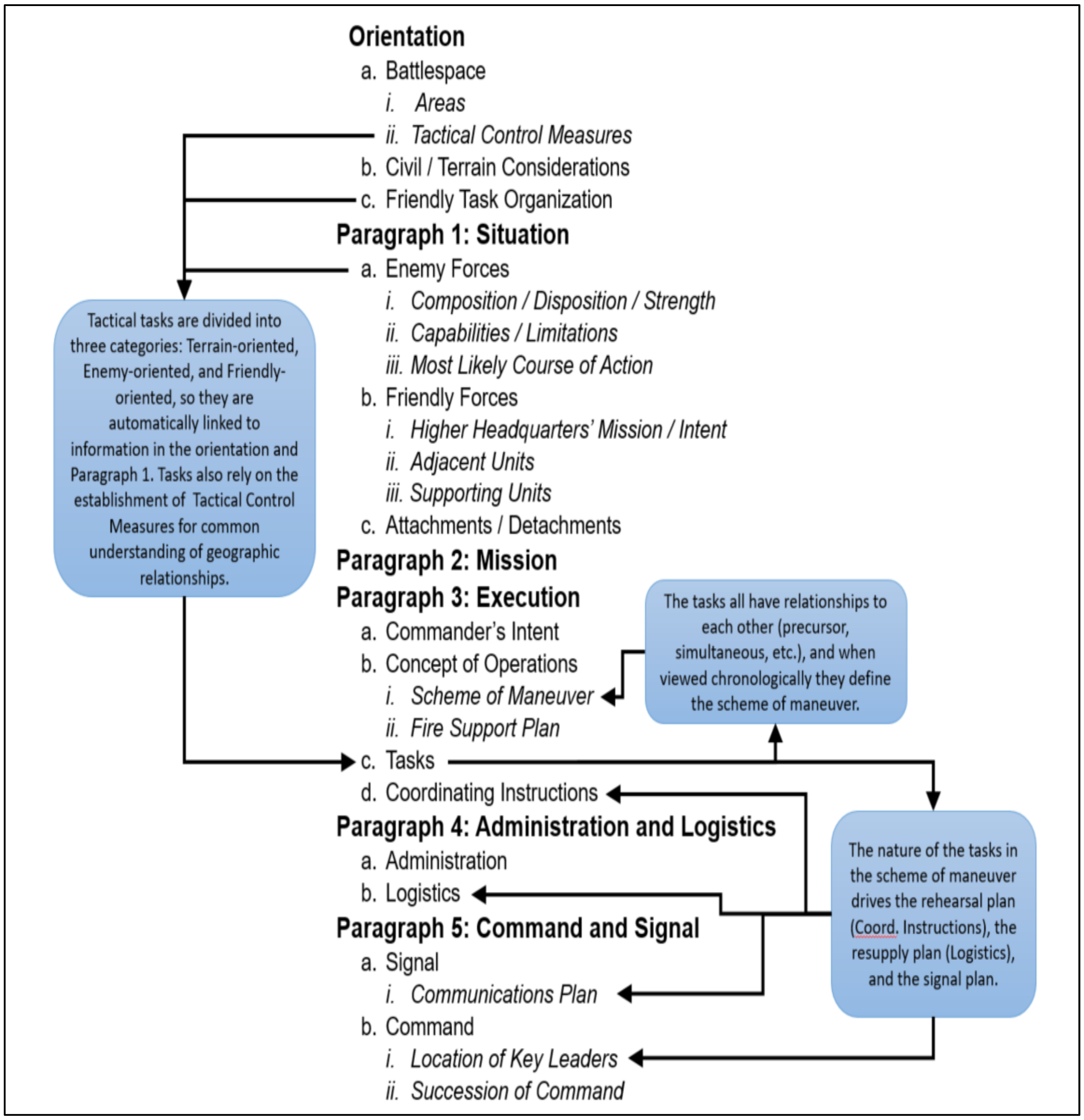

Orientation
a. Battlespace
i. Areas
ii. Tactical Control Measures
b. Civil / Terrain Considerations
c. Friendly Task Organization
Paragraph 1: Situation
a. Enemy Forces
i. Composition / Disposition / Strength
ii. Capabilities / Limitations
iii. Most Likely Course of Action
b. Friendly Forces
i. Higher Headquarters' Mission / Intent
ii. Adjacent Units
iii. Supporting Units
c. Attachments / Detachments
Paragraph 2: Mission
Paragraph 3: Execution
a. Commander's Intent
b. Concept of Operations
i. Scheme of Maneuver
ii. Fire Support Plan
c. Tasks
d. Coordinating Instructions
Paragraph 4: Administration and Logistics
a. Administration
b. Logistics
Paragraph 5: Command and Signal
a. Signal
i. Communications Plan
b. Command
i. Location of Key Leaders
ii. Succession of Command
Tactical tasks are divided into three categories: Terrain-oriented, Enemy-oriented, and Friendly-oriented, so they are automatically linked to information in the orientation and Paragraph 1. Tasks also rely on the establishment of Tactical Control Measures for common understanding of geographic relationships.
The tasks all have relationships to each other (precursor, simultaneous, etc.), and when viewed chronologically they define the scheme of maneuver.
The nature of the tasks in the scheme of maneuver drives the rehearsal plan (Coord. Instructions), the resupply plan (Logistics), and the signal plan.

Figure 2. Example of Internal Order Information Dependencies – Tactical Tasks.

In normal, manual use, every one of these interconnections is a potential point of error. If a leader changes a task in Paragraph 3 but neglects to change the associated tactical control measures or the signal plan the result could be confusion on the battlefield. The more complex the order the greater the potential for disconnects between the different paragraphs if the person writing the order is manually attempting to keep everything straight.

In an informal review of several operations orders used in training missions during pre-deployment workups, the authors found that as much as 80% of the information in a given operations order is tied in some way to other information in that same order or to respective orders of adjacent units or higher headquarters.

c. External Dependencies

External connections between the information in a given operations order and those of higher headquarters or adjacent units pose the same issues as internal dependencies, with the added complication that multiple leaders are often working on their orders simultaneously. Additionally, there is a lot of information in a given unit's order that is drawn directly from the order published by its parent unit. In a manual, orders-generation process, that information must be manually copied verbatim from one order to the other. This is an inefficient use of a leaders' time and is an example of the format driving the process.

Other parts of the order are dependent on external stores of information, or inform additional external processes and products. Some examples include TCMs that are generated by higher headquarters and are propagated to all units via tactical overlays, information about the enemy situation that comes in a separate intelligence update, and information about friendly unit locations and the capabilities of supporting fires agencies that come from other command and control systems. Much of this information is copied over and over again from order to order with little variation, another example of wasted effort and energy that could instead be focused on the tactics of each specific plan. Examples of this type of external dependency are depicted in Figure 3.

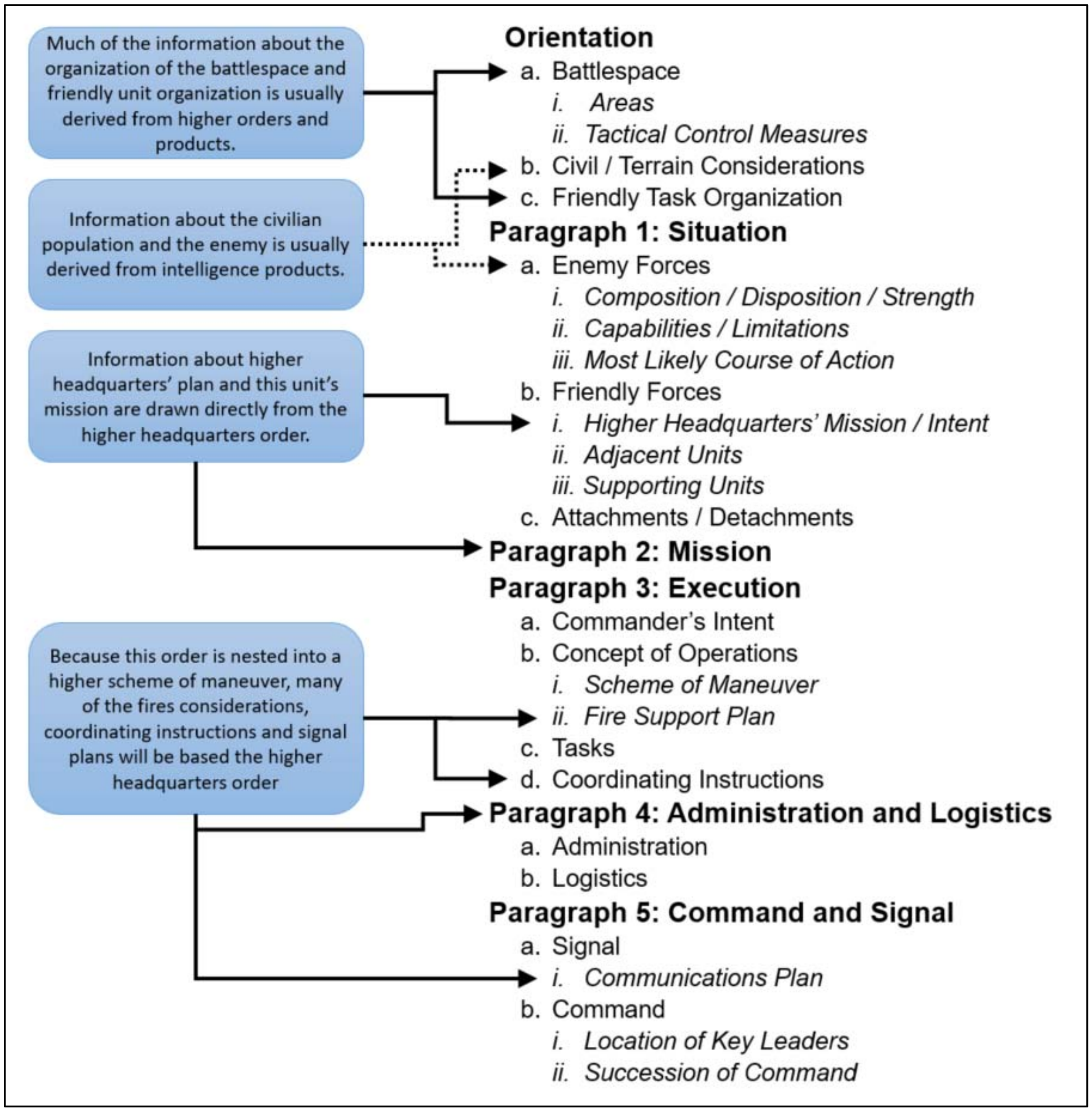
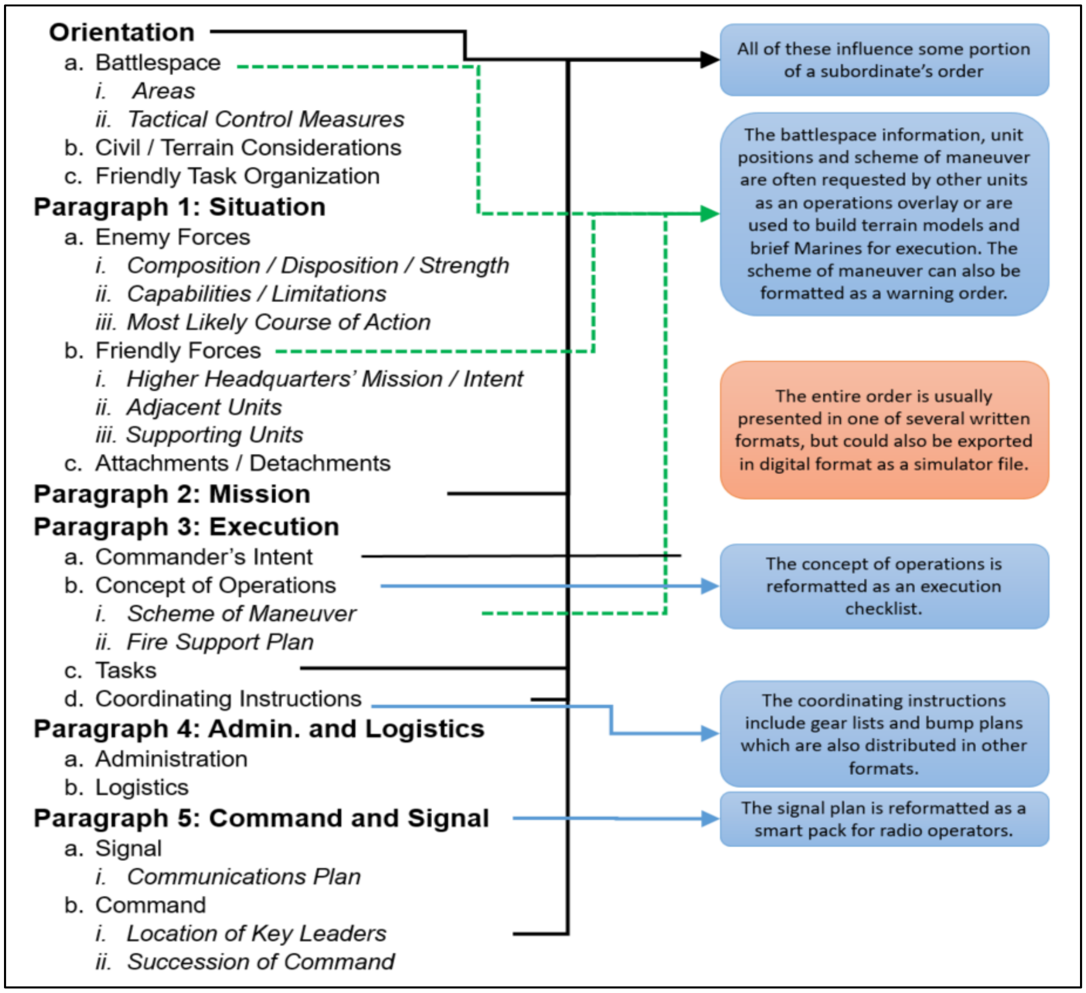

Much of the information about the organization of the battlespace and friendly unit organization is usually derived from higher orders and products.
Information about the civilian population and the enemy is usually derived from intelligence products.
Information about higher headquarters' plan and this unit's mission are drawn directly from the higher headquarters order.
Because this order is nested into a higher scheme of maneuver, many of the fires considerations, coordinating instructions and signal plans will be based the higher headquarters order
Orientation
a. Battlespace
i. Areas
ii. Tactical Control Measures
b. Civil / Terrain Considerations
c. Friendly Task Organization
Paragraph 1: Situation
a. Enemy Forces
i. Composition / Disposition / Strength
ii. Capabilities / Limitations
iii. Most Likely Course of Action
b. Friendly Forces
i. Higher Headquarters' Mission / Intent
ii. Adjacent Units
iii. Supporting Units
c. Attachments / Detachments
Paragraph 2: Mission
Paragraph 3: Execution
a. Commander's Intent
b. Concept of Operations
i. Scheme of Maneuver
ii. Fire Support Plan
c. Tasks
d. Coordinating Instructions
Paragraph 4: Administration and Logistics
a. Administration
b. Logistics
Paragraph 5: Command and Signal
a. Signal
i. Communications Plan
b. Command
i. Location of Key Leaders
ii. Succession of Command

Figure 3. Examples of External Information Dependencies in the Operations Order.

There are also outputs that are generated from a given operations order that inform other units and processes. These also fall into the category of external dependencies. For example, a leader needs to communicate the resulting fire support plan (usually in a different format from the one used in the order) to any external agencies supporting his or her maneuver. Small-unit leaders are given equipment lists and plans for rehearsal and inspections based on the tasks that are being accomplished. If the operation uses vehicles, there are separate manifests, load plans or “bump” plans to handle the organization of loading/unloading or to handle the loss of a vehicle. All of these special outputs and formats are simply a restructuring of information that is already in other parts of the order, but building them manually places an additional burden on the leader writing the order. Examples of this type of external dependency are depicted in Figure 4.



Orientation
a. Battlespace
i. Areas
ii. Tactical Control Measures
b. Civil / Terrain Considerations
c. Friendly Task Organization
Paragraph 1: Situation
a. Enemy Forces
i. Composition / Disposition / Strength
ii. Capabilities / Limitations
iii. Most Likely Course of Action
b. Friendly Forces
i. Higher Headquarters' Mission / Intent
ii. Adjacent Units
iii. Supporting Units
c. Attachments / Detachments
Paragraph 2: Mission
Paragraph 3: Execution
a. Commander's Intent
b. Concept of Operations
i. Scheme of Maneuver
ii. Fire Support Plan
c. Tasks
d. Coordinating Instructions
Paragraph 4: Admin. and Logistics
a. Administration
b. Logistics
Paragraph 5: Command and Signal
a. Signal
i. Communications Plan
b. Command
i. Location of Key Leaders
ii. Succession of Command
All of these influence some portion of a subordinate's order
The battlespace information, unit positions and scheme of maneuver are often requested by other units as an operations overlay or are used to build terrain models and brief Marines for execution. The scheme of maneuver can also be formatted as a warning order.
The entire order is usually presented in one of several written formats, but could also be exported in digital format as a simulator file.
The concept of operations is reformatted as an execution checklist.
The coordinating instructions include gear lists and bump plans which are also distributed in other formats.
The signal plan is reformatted as a smart pack for radio operators.

Figure 4. Examples of External Products Derived from the Operations Order.

B. ORDER CREATION AND DISSEMINATION

1. Doctrine

The Marine Corps uses an operation planning process that is aptly named the Marine Corps Planning Process (MCPP), detailed in Marine Corps Warfighting Publication 5-1 (United States Marine Corps 2010). This process is scalable to fit the size of the operation being planned, from company-level tactics to entire campaign plans. It consists of six phases arranged in a cycle, starting with Problem Framing (Figure 5).

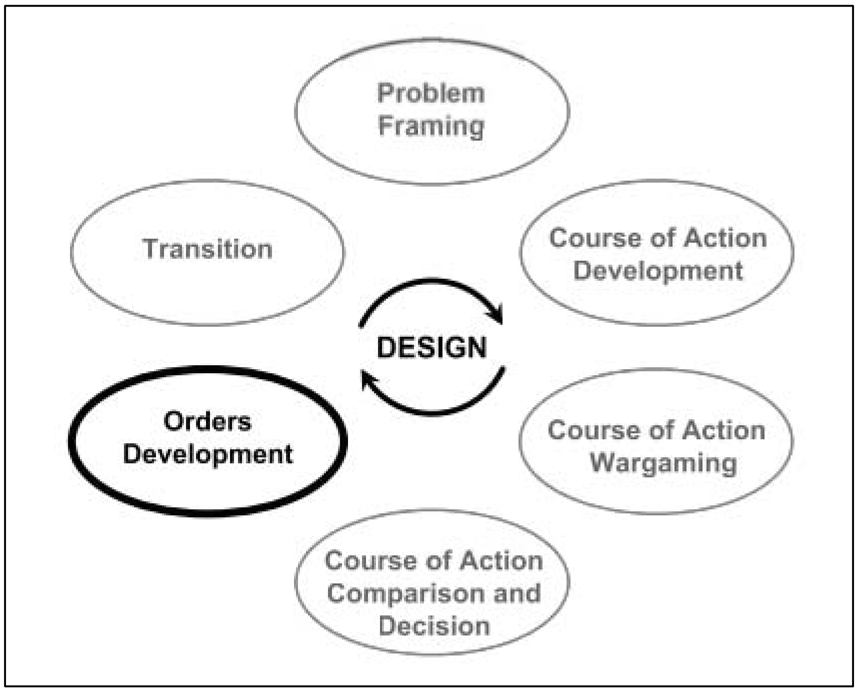

Problem Framing
Course of Action Development
Course of Action Wargaming
Course of Action Comparison and Decision
Orders Development
Transition
DESIGN

Figure 5. The Marine Corps Planning Process. Source: United States Marine Corps (2010).

Problem Framing consists of organizing all of the information available about a given battlespace and seeking to understand the problem that the operation must solve. Course of Action (COA) Development, COA Wargaming, and COA Comparison/Decision all revolve around choosing a specific plan toward which to commit the unit’s resources. The actual process of capturing the plan in a communicable format (Orders Development) should only occur once the idea for the plan has been formulated. Finally, Transition is the step where action is taken on an order—whether briefing it to Marines for execution or passing it from a future operations cell to a current operations cell for further refinement. The cyclical nature of the process seeks to ensure that if new ideas or information come to light they can be incorporated into the broad understanding of the operation and COA formulations and decisions can be revisited as necessary (United States Marine Corps 2010).

This same process is theoretically applied at the tactical level, including the ability to adapt an order mid-development if new information comes to light (The Basic School 2015a). Unlike planning at the battalion level or higher, however, leaders at the lower tactical levels are often generating orders alone, without the staff personnel needed to quickly rework a plan administratively if need be. Their timelines are usually more compressed than those experienced by leaders planning at the battalion level or above (it is common for a battalion to develop the plan a single operation for several weeks, whereas units at the company level often have less than a day—or even less than an hour—to formulate and brief a plan). Furthermore, they usually do not have the luxury of waiting until their ideas are finalized to begin capturing them in a communicable format—for the sake of their Marines they usually issue a "Warning Order" containing the information needed to begin preparations as soon as the basics of the plan are sketched out. The result of all of these factors is a "bird in the hand" mentality: small unit leaders can be reluctant to change a plan once they have started writing it down and can often be said to have "fallen in love" with their plans.

The Marine Corps' doctrine does not specify or prescribe a format for the completed operations order, and acknowledges that they may even be purely verbal (United States Marine Corps 2010), but the reality is that Marines default to what they know, and all Marine leaders know the five paragraph format.

2. Best Practices

From the outset of training at TBS, new officers are told that although they will receive a solid grounding in the fundamentals of the tactical order writing process, their understanding of how to quickly sum up a situation, formulate a plan and communicate that plan effectively will come with repeated practice and experience (The Basic School 2015a). Much of that understanding comes in the form of "best practices"—ideas and methods gleaned from the experiences and instructions of others. This begins at TBS with the instruction provided by each student's Staff Platoon Commander and continues with the feedback and personal anecdotes of other staff members who coach the students through a series of field exercises.

Once assigned to the operating forces, leaders continue to hone their abilities, to include order writing, through their own successes and failures and also through their initiative to learn from peers and more experienced leaders. This can be done through a number of modes ranging from personal interaction to reading the opinions of others expressed in books or publications such as the Marine Corps Gazette. The end goal is a commander who possesses "coup d'oeil" ("stroke of the eye"), defined by Carl Von Clausewitz (1989) as the ability to

> see things simply, to identify the whole business of war completely with himself, that is the essence of good generalship. Only if the mind works in this comprehensive fashion can it achieve the freedom it needs to dominate events and not be dominated by them. (578)

With very little of the instruction in proper tactical thought and communication formalized beyond entry level training, it is easy to see how individuals are largely the products of their personal initiative and circumstances (to include the other leaders to whose influence they are exposed). It is the author's assertion based on experience teaching new Marine Corps officers that one method of enhancing this largely oral tradition would be to develop a tool that incorporates best practices in tactical thought and coaches less experienced users in the application of those best practices.

3. Pitfalls and Shortfalls

The Five Paragraph Order format as it exists today does a fine job of conveying information to Marines in a manner that they are trained to digest, but the process of preparing orders is fraught with problems. There are several broad issues with the current format and its associated orders generation process that can be alleviated with a software solution while simultaneously preserving the familiar structure, and therefore not invalidating years of training methodology.

a. One Size Fits All

It is the experience of the author during time spent teaching operations orders at The Basic School that the emphasis placed on proper orders format during entry-level training is a necessary evil. Providing student lieutenants with a standard way of organizing and communicating their thoughts is immensely valuable. This is undoubtedly one of the reasons that the format has grown in size and prescriptiveness over time—whenever someone comes up with a good idea for communicating a portion of an order, that idea is incorporated as part of the "standard." Thus, for example, the counter-insurgency efforts in Iraq and Afghanistan prompted the addition of cultural and geo-political information into the situation paragraph of the order (Department of the Army 2014). While these issues were appropriate to consider and brief in counterinsurgency fights, the danger is that expanding the operations order format to fit those details added a permanent layer of complication even for situations in which that information has little relevance.

This expansion of the orders format to fit all possible contingencies has a distinct and observable downside of causing inexperienced leaders to adjust their plan to fit the format rather than adjusting the format to fit their plan. This problem could be remedied by a method that forces the user to formulate a focused plan and then converts that plan into written documents and templates as required.

b. Redundancy and Error

As we have previously demonstrated, the basic structure of the Five Paragraph Order contains myriad internal and external dependencies, with the same information being formatted in different ways for different audiences. While this makes it easy for subordinates who are familiar with the format to find the specific pieces of information that they need, it also introduces an un-necessary amount of complexity to the process of actually writing an operations order. The natural result of the interdependencies is that a change made in one part of the order has a “ripple effect” throughout the rest of the order. The more pronounced the change, the greater the ripples. The author observed in the course of grading hundreds of student orders during his time as an instructor at TBS that most errors are the result of un-reconciled changes made to a different portion of the order.

Some errors in tactical orders are inconsequential. Others are obvious to anyone who spots them and are summarily dismissed. Due to the nature of the business of war, however, there is a significant chance that an error in an operations order could not only be gravely consequential, but also could be difficult to identify as an error due to the stress and fatigue of combat. This is especially true given the short planning timelines associated with orders generation at the leading edges of the battlefield. A system that reduces the probability of these errors occurring would be greatly beneficial to the armed services.

c. Combat and Human Factors

The redundant and error-prone nature of the orders format is magnified by the fact that its primary purpose is to serve as a communication vehicle during the stress of combat, an activity that is inherently chaotic and distracting. It is widely understood that stress and fatigue effect decision making, but what is less studied is the role that the format of the operations order plays in enhancing or mitigating the effects of that stress.

Neuroscience and Behavioral Reviews published a review in 2012 summarizing nearly 30 years of scientific studies on the effects of stress (Starcke and Brand 2012). Among its findings are some that directly relate to the decision-making abilities of leaders under significant stress from fatigue, fear, uncertainty, anger, grief, and other emotions that commonly accompany combat operations. The review found fairly unanimous agreement among the scientific community on three common responses to stress that are relevant to the military decision maker:

(1) "Dysfunctional strategy use" – People under unusual amounts of stress do a poor job of evaluating all of their available options, and tend to decide on an outcome prematurely—that is, they choose before they know all of the necessary information. (Starcke and Brand 2012, 1234).

(2) "Insufficient adjustment from automatic response" – The surveyed research was mixed on whether people tend toward more or less risky decisions under stress, and seemed to depend on personalities of those being tested and the reward schemes used by the researchers. The studies were unanimous, however, in confirming that stress increases the "bounding effect"—the idea that decisions are largely biased by the context in which they are originally presented. (1236).

(3) "Altered feedback processing" – People under stress tend to rely more on the immediately available opinions of those around them, regardless of the correctness of those opinions. (1236).

These findings reinforce the idea that manual creation of the current orders process under stress is prone to error. The current process is based more on format than it is on content, and therefore it does not lead a user to focus on available options. It is also prone to the bounding effect in that users are more likely to copy and paste ideas from higher headquarters' order without applying critical thought to correctness. A system that helps the user to focus on organizing necessary tasks and simultaneously checks for correctness might do a lot to mitigate the effects of stress on the order generation process.

C. EXISTING TECHNOLOGY

The concept of using computers to aid in planning is not new, nor is the idea of using data structures or mobile technology to make complicated tasks simpler for the user. In order to understand how an application like TOOTH might fill the current technology gap, it is necessary to discuss the significant work that has already been done in the area of planning tools and applications.

1. Military

Given that planning and conducting operations is the essence of the military's existence, it is not surprising that there are a number of tools and processes dedicated to collecting, organizing, sharing, and displaying tactical information. What is surprising is that, for all of the effort spent capturing the "how" of military operations, there has been very little technical emphasis placed on improving the thought process of the junior leader regarding the "what"—helping them to build and communicate sound plans.

The following are some of the tools available in the military today, along with reasons why each presents only a part of a possible holistic solution:

a. Languages and Data Structures

The idea of defining a data structure that captures all of the interactions between pieces of information in an operations order is not new. There is NPS thesis work dating back to 1994 that attempted to organize tasks for output in a now-defunct set of simulator languages (Mohn 1994; Serbest 1994).

A more lasting effort has been to codify military terminology in a way that allows universal communication between various U.S. military and allied C2 and simulator systems. Battle Management Language (BML) sought to define a precise language for tasking and reporting that would capture doctrinally correct tasks and other order information without constraining the free expression of commander's intent (Pullen, Hieb, and Levine 2007).

The BML effort became a North Atlantic Treaty Organization (NATO) effort in September of 2004, with the acknowledgement that “an open framework is needed to establish coherence between C2 and Modeling and Simulation (MaS) type systems in order to provide automatic and rapid unambiguous initialization and control of one by the other” (NATO Research and Technology Organization 2012, 1–2). In total, more than nine separate systems and languages have been integrated into a very comprehensive set of definitions for the interaction between systems (Pullen, Hieb, and Levine 2007). The results of the initial BML effort were released in 2012 with the recommendations for continued work to bring the technology to a readiness level that would support operational use (NATO Research and Technology Organization 2012).

BML as a structure is very robust, but has been developed from a perspective of system-to-system communication. Thus, many of the aspects of an operations order that are derived directly from information such as task relationships or commander’s intent are expressed merely as “free text” (Pullen et al. 2011). Furthermore, BML does not appear to support finely-grained relationships between tasks of the type necessary to describe a scenario in which one task had a pre-determined overlap with another, or where one task had to both start after another task and also end before it (nested inside of said task). This issue will be addressed again in Paragraph III, Application Design.

b. Planning / C2 Technologies

The DOD uses numerous desktop computing systems and software for those systems that assist in timely command and control of friendly forces. These systems contain very robust capabilities for graphically planning operations and real-time tracking of movement and reporting information. Among these systems are Northrup Grumman’s Command and Control Personal Computer (C2PC) (Figure 6), the Command Post of the Future (CPOF), and the Joint Military Planning System–Expeditionary (JMPS-E).

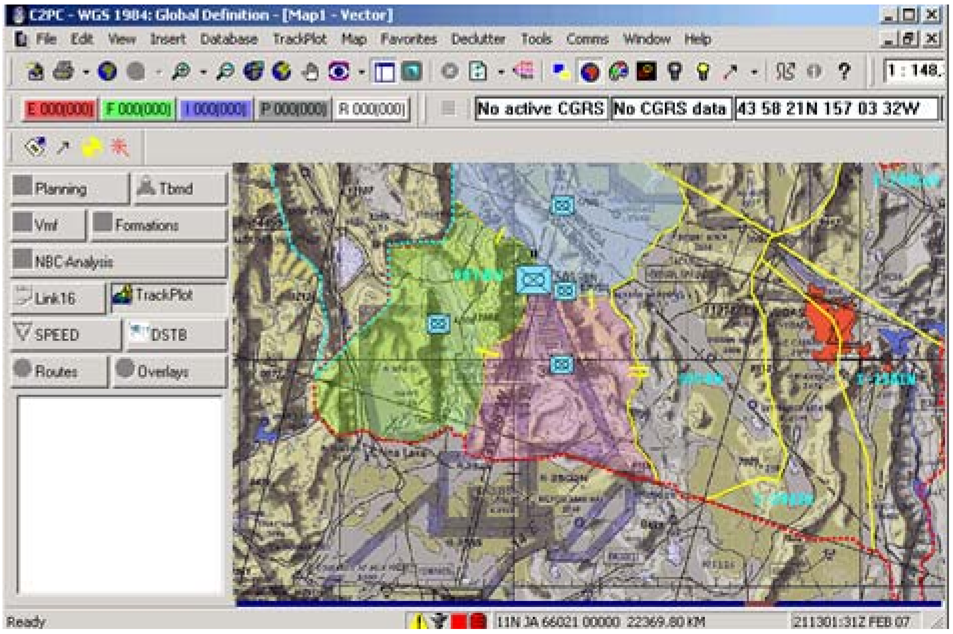

C2PC - WGS 1984: Global Definition - [Map1 - Vector]
File Edit View Insert Database TrackPlot Map Favorites Declutter Tools Comms Window Help
No active CGRS
No CGRS data
43 58 21N 157 03 32W
Planning
Tbmd
Vmf
Formations
NBC Analysis
Link16
TrackPlot
SPEED
DSTB
Routes
Overlays
Ready
11N JA 66021 00000 22369.80 KM
211301:31Z FEB 07

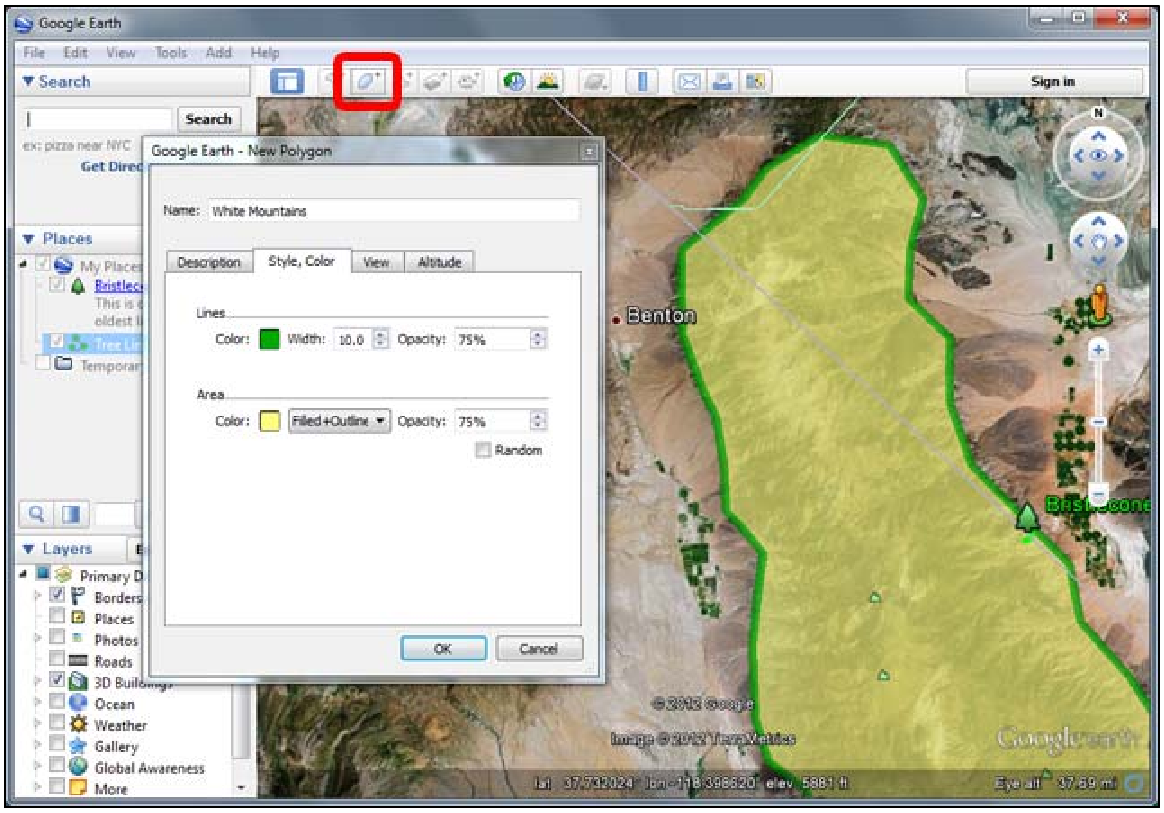
Figure 6. C2PC Screen Capture. Source: tandef (2012).

While robust, these systems are focused on mapping and reporting and place little to no emphasis on tying information together as a holistic plan. Furthermore, these systems are not widely available in a mobile format and require significant levels of operator training in order to be used effectively, even in a desktop Combat Operations Center (COC) environment.

c. Mapping Technologies

The military also uses robust mapping software in ways that are an extension of their original purpose. One example of this is the use of the National Geospatial-Intelligence Agency's FalconView, the mapping software that makes up the core of the Air Force's Portable Flight Planning Software (NGA 2016). The mapping aspect of FalconView looks very similar to that of C2PC, but the ease of loading satellite images has long made it a favorite tool of the ground operations community. The author first used it in Iraq in 2005, and it was still being taught at the Expeditionary Warfare School (EWS) when he attended in 2012. Another example of robust mapping technology is Google's GoogleEarth (Figure 7), which was introduced to the author as a planning tool during his training at the

Tactical Marine Air Ground Task Force (MAGTF) Integration Course (TMIC), and that he observed in frequent use as an alternate situational awareness tool in COC environments both in training and in Afghanistan. It is attractive as a tool in part because of the simplicity of its learning curve and its ability to quickly represent three dimensional shapes, which are useful when dealing with fire support TCMs.



Name: White Mountains

Lines
Color: 🟩 Width: 10.0 Opacity: 75%

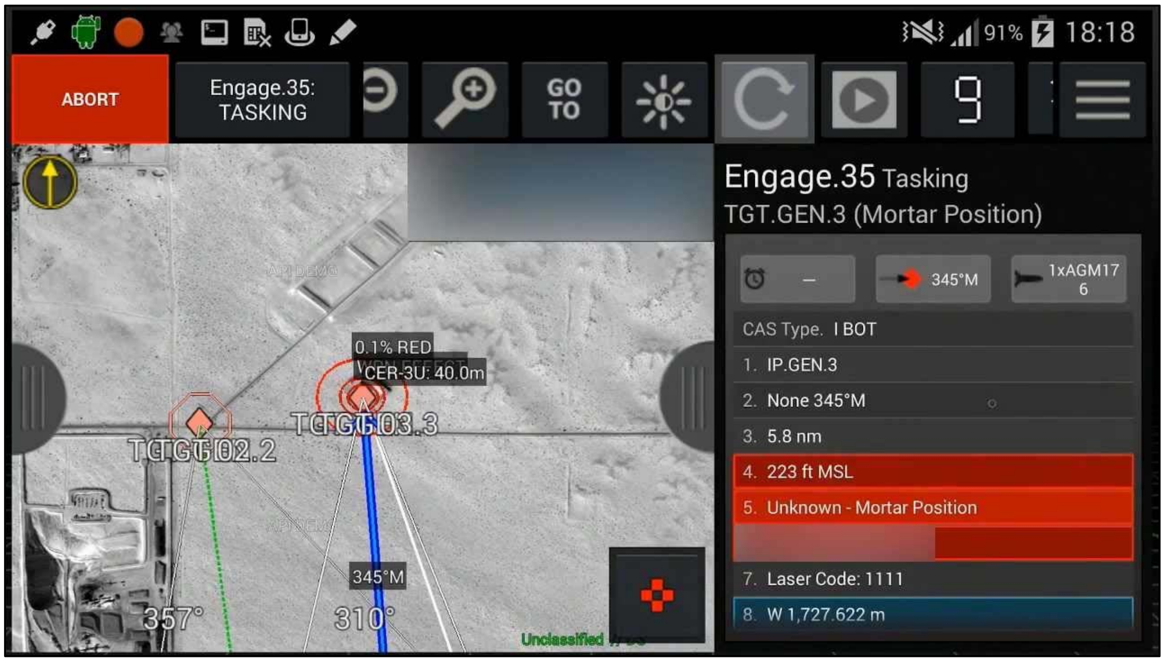
Area
Color: 🟨 Filled+Outline Opacity: 75%
☐ Random

Figure 7. Google Earth Screen Capture. Source: Google Earth.

While very useful for mapping the special relationships between information, this type of software has no capability of storing and organizing the other information inherent to the operations order.

d. Mobile Technologies

The military has already begun embracing mobile technology, with some services taking a more active role in the acquisition and fielding of mobile devices and applications than others. The Army has established a mobile application store and a branch in its doctrine organization dedicated to vetting and publishing applications for soldier use (Lopes 2015). Without a similarly established capability, the Marine Corps is arguably behind in this regard, but it does have at least one very impressive tool in KILSWITCH (see Figure 8), part of NAVAIR's Electronic Knee Board initiative (NAVAIR 2015). This software was used by the author extensively in Afghanistan. It provides a handheld GPS-enabled mapping tool capable of displaying satellite imagery and grid reference graphics (GRGs) and on which the user can add their own overlay information. In this capacity, it acts as a sort of "poor man's C2PC" for the mobile environment. It also has significant capabilities in its primary mission as a fires planning tool, to include integration of full-motion video from support assets.



Engage.35 Tasking
TGT.GEN.3 (Mortar Position)

– 345°M 1xAGM176

CAS Type. I BOT
1. IP.GEN.3
2. None 345°M
3. 5.8 nm
4. 223 ft MSL
5. Unknown - Mortar Position
7. Laser Code: 1111
8. W 1,727.622 m

Figure 8. KILSWITCH Screen Capture. Source: NAVAIR (2015).

While incredibly useful for mapping elements of an operations order, *KILSWITCH* also mirrors more robust desktop software in its lack of processes for handling the majority of the information resident in operations orders.

e. Expert Systems

The increase in accessible computing power in the 1980s led to an explosion in popularity of a subset of artificial intelligence (AI) technology known as "expert systems," computer algorithms that "attempt to duplicate results obtained by *actual* experts in a particular field or domain" (Franklin et al. 1988, 1328). These systems are typically built by programmers who use logic-based languages such as Prolog to encode the thought process of recognized experts in a given field. This usually consists of a series of in depth interviews translated into logical constructs ("if this is true, then that is true"). The resulting knowledge base of facts could then be leveraged by personnel without the same level of training and experience (Franklin et al. 1988). Figure 9 gives a diagram of a typical expert system.

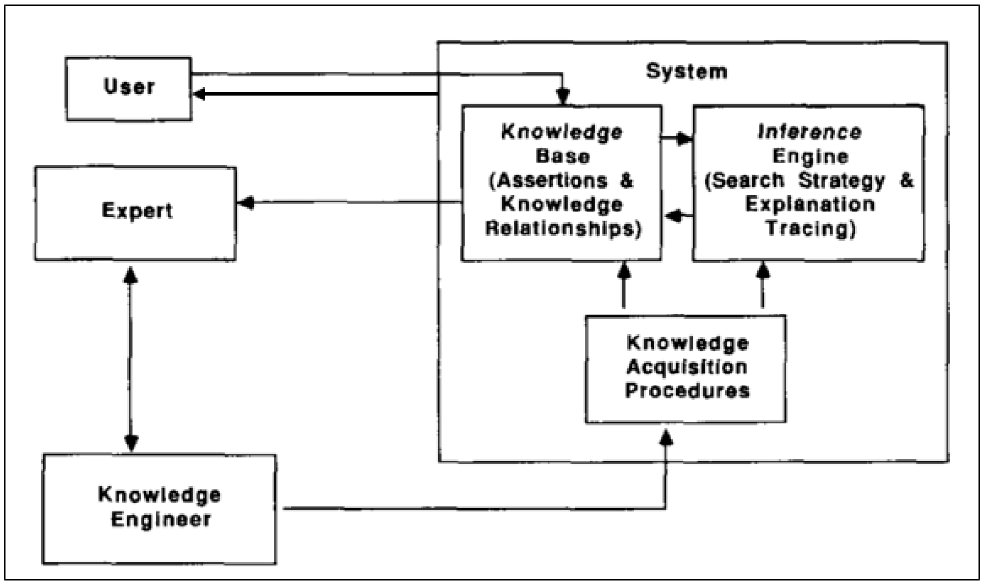

User
System
Knowledge Base (Assertions & Knowledge Relationships)
Inference Engine (Search Strategy & Explanation Tracing)
Expert
Knowledge Acquisition Procedures
Knowledge Engineer

Figure 9. Diagram of a Typical Expert System. Adapted from Franklin et al. (1988).

This methodology was originally applied to every conceivable aspect of the military, from recognition and categorization of enemy vehicles to truck repair, and everything in between, including operational planning. Within a decade, however, the DOD seemed to have soured on the concept of an expert system being able to effectively plan complex military operations—the number of input variables was simply too large. In her work on naturalistic decision-making, originally published in 1997, Caroline E. Zsambok concludes that

> battle command is one of the most complex decision-making environments in which true expertise comes into play. Unlike games like chess or physics problems, battles go on far longer, the forces are not equal, there are external forces acting, and sometimes it is difficult to know whether you won or lost. The definition and measurement of expertise, therefore, are quite difficult. Experience and training, perhaps, are the best measure of expertise. (Zsambok and Klein 2014, 77)

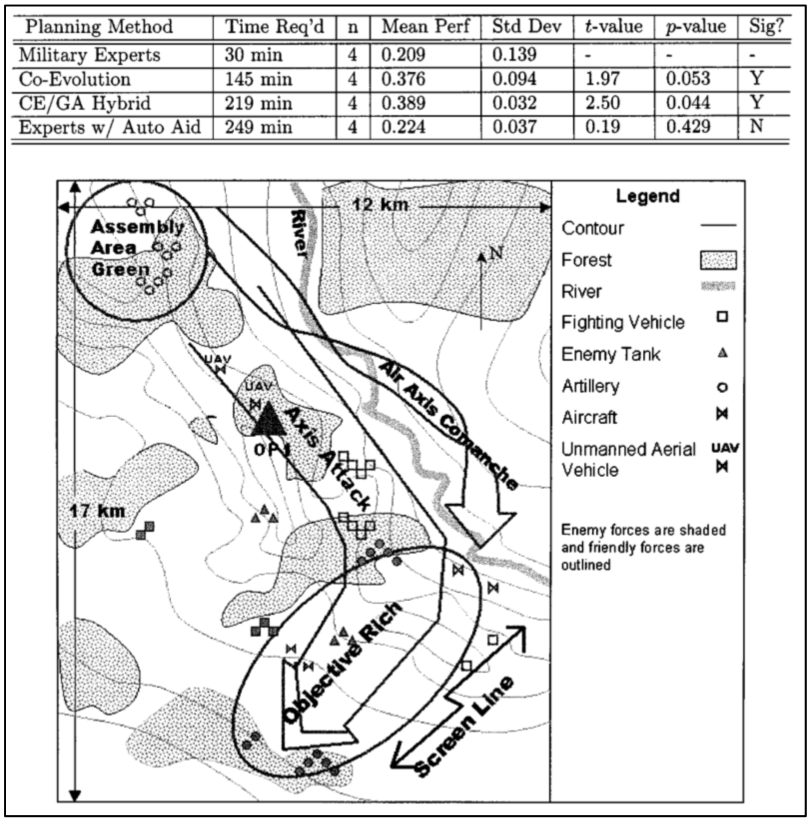
That is not to say, however, that the DOD gave up on the use of AI. Military researchers instead began to focus on newer fields of AI as a means to improving military planning. One such experiment is detailed in a 2002 paper by Robert Kewley and Mark Embrechts. Their approach was to use a type of computational system called "fuzzy-genetic decision optimization" (FGDO) in combination with more traditional probabilistic military simulations to test whether a computer could arrive at a more optimal planning solution than a human military expert for a given scenario (Kewley and Embrechts 2002). Their system took the following approach to generating plans:

1. A military commander's preferences for the end result of a given tactical action were entered into the software in order to determine a numerical set of target outcomes.
2. A genetic algorithm produced a set of battle plans covering iterations within a range of possible options.
3. Those battle plans were fed into a traditional stochastic combat simulation.
4. A preference model examined the results of each simulation and assigned that plan a score based on its ability to meet the commander's intent.

5. The genetic algorithm added that result to its information base and generated the next set of plans to test.

6. This loop continued until halted, producing plans that scored better and better in the simulator over time, eventually eclipsing the results of plans produced by military planning experts (Kewley and Embrechts 2002).

The test ultimately compared six different sets of plans: Those produced by the military experts, those produced by four different versions of their algorithms, and a set of computer-generated plans modified by military experts. Given enough time, the computer algorithms were able to come up with plans that scored better than the best human plans in the simulator every time (Figure 10). However, the computer in this scenario took up to eight times as long as the human expert to achieve this result.



Planning Method	Time Req'd	n	Mean Perf	Std Dev	t-value	p-value	Sig?
Military Experts	30 min	4	0.209	0.139	-	-	-
Co-Evolution	145 min	4	0.376	0.094	1.97	0.053	Y
CE/GA Hybrid	219 min	4	0.389	0.032	2.50	0.044	Y
Experts w/ Auto Aid	249 min	4	0.224	0.037	0.19	0.429	N

Figure 10. Example Results and Scheme of Maneuver (SOM) for AI Mission Planning. Source: Kewley and Embrechts (2002).

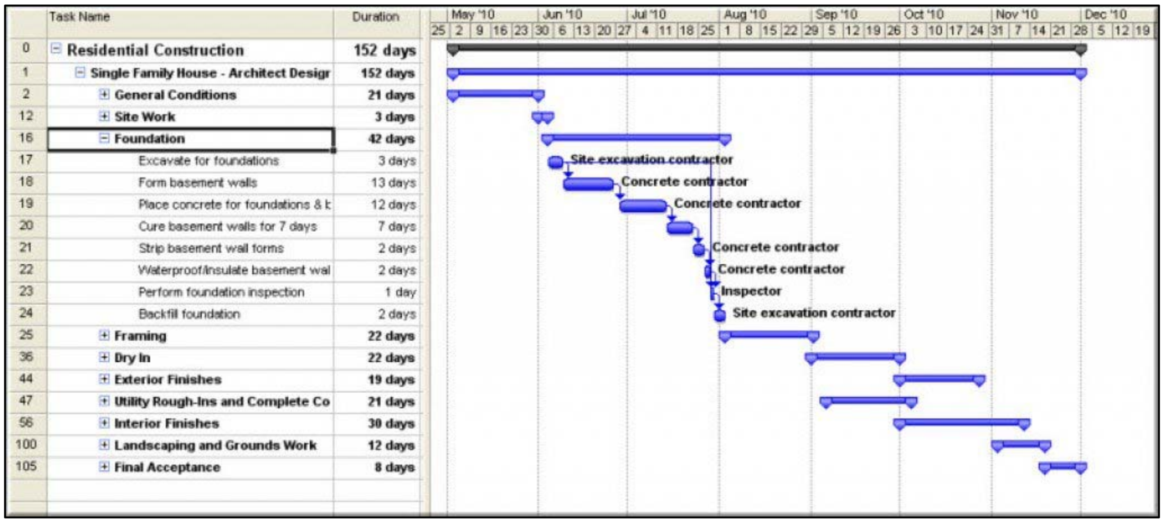
Despite the relatively-advanced, good performance of this system and the subsequent increase in technology availability, there is no system available in a distributed tactical environment that helps a commander by suggesting possible courses of action. However, the system also showed that applying human decision-making to algorithmic thought could improve the overall simulated outcome. If a similar algorithmic power was instead dedicated to helping make a commander's decision-making more efficient in the first place, such a system could leverage both the logical ability of the computer with the obvious speed advantage of a trained military mind.

2. Civilian

The military is not unique on the need to plan large and complicated operations. With the exception of interpersonal violence, many military processes have a direct correlation to the civilian sector. Thus, it is no surprise that there is a significant amount of civilian research devoted to the application of technology to efficient planning and communication. For example, computer systems are increasingly being used to plan and optimize large construction projects (Hartmann et al. 2012).

a. Planning Technologies

Another analogy to be made between military planning and the civilian sector is the similarity between detailing the many concurrent efforts of a military campaign and the plan to develop, test, and release a product in the business world. Collaborative planning software, such as Microsoft Project, has long helped to visually depict the relations between tasks and events through the use of automated Gantt charting (Figure 11).



	Task Name	Duration
0	Residential Construction	152 days
1	Single Family House - Architect Desigr	152 days
2	General Conditions	21 days
12	Site Work	3 days
16	Foundation	42 days
17	Excavate for foundations	3 days
18	Form basement walls	13 days
19	Place concrete for foundations & k	12 days
20	Cure basement walls for 7 days	7 days
21	Strip basement wall forms	2 days
22	Waterproof/insulate basement wal	2 days
23	Perform foundation inspection	1 day
24	Backfill foundation	2 days
25	Framing	22 days
36	Dry In	22 days
44	Exterior Finishes	19 days
47	Utility Rough-Ins and Complete Co	21 days
56	Interior Finishes	30 days
100	Landscaping and Grounds Work	12 days
105	Final Acceptance	8 days

Figure 11. Microsoft Project Gantt Chart Example. Source: Anderson (2010).

Although the Program Evaluation and Review Technique (PERT) and its subsequent business strategies originally grew out of military programs (Weaver 2007), the divergence between military and civilian planning needs means that most civilian tools are optimized toward specific use cases and no longer transfer well to military processes. Furthermore, there is a more granular need for mapping and real-time reporting in the military than is generally needed in the civilian world, and the software differences bear that out.

b. Mapping Technologies

The rapid advances in mobile computing have led to the creation of a robust set of mapping tools that are available to developers for inclusion into their applications. The scalability of these applications when tied to a cloud infrastructure has led to some novel uses on the part of individuals and businesses (Miller 2006). Google's GoogleMaps is one of the most prevalent examples of this software (another is Microsoft's Bing Maps*)*. GoogleMaps allows a person to use the GPS in their device to plot their location on a number of different map layers, and allows application designers to populate the map with customized icons based on everything from user preferences to search terms to context-based content.

OpenStreetMap (OSM) attempts to provide a similar service, but without the proprietary data restrictions that companies like Google and Microsoft can impose on their products (Buczkowski 2015). The software code and all of the data is open-sourced, to include contributions by a vibrant community of editors. While Google strives to provide users with a sleek interface that highlights only what it thinks the user wants to see, the OSM software lacks the same clean feeling (Figure 12). The accessibility of OSM has led to the formation of an active user community, however, and in some areas of the world its maps are more accurate and useful than those produced by the major corporations (Cipeluch et al. 2010).

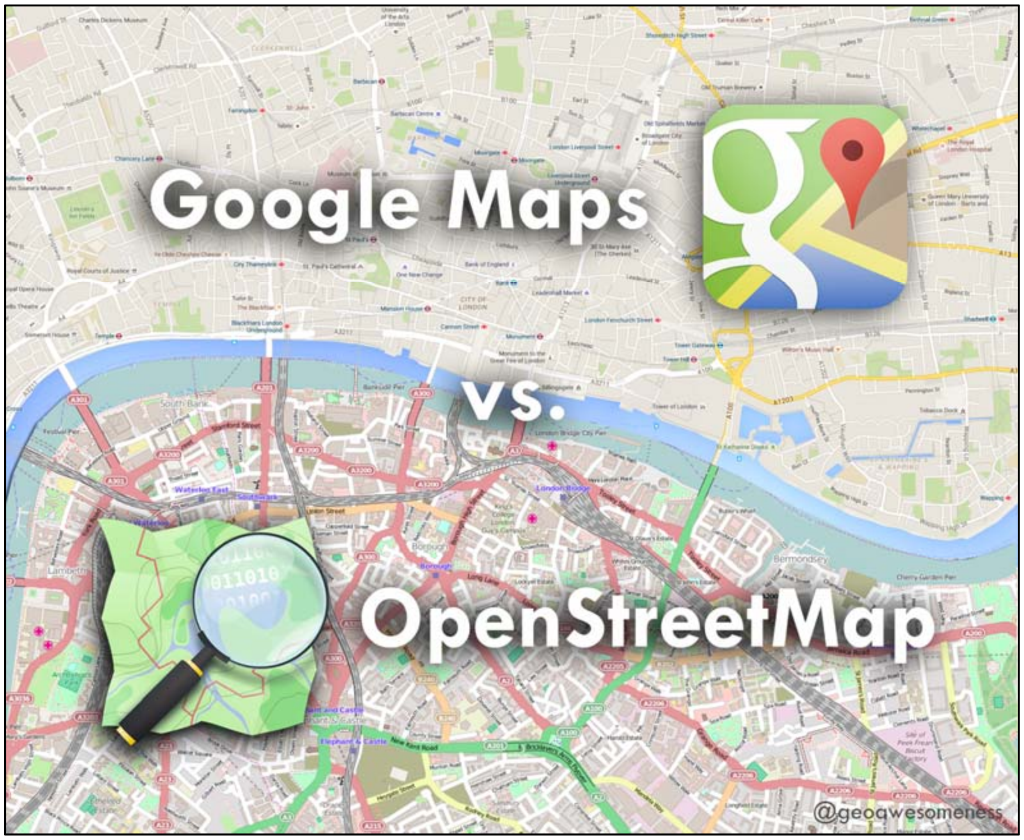


Figure 12. Google Maps vs. OpenStreetMap Screen Captures. Source: Buczkowski (2015).

Another relatively new contender in the mapping application space is WorldWind, a National Air and Space Administration (NASA) project that provides open-source access to its code. The aim of WorldWind is to allow researchers and application developers to harness the power of NASA's imagery and technology to improve communications and education (NASA 2017).

c. Artificial Intelligence and Expert Systems

When people think artificial intelligence (AI), one of the first names to pop into their heads is IBM's Watson, the supercomputer AI that beat human players in the television game show *Jeopardy* in 2011. Fewer people know of Watson's subsequent task as an intelligent advisor to medical professionals, but that type of application is what led IBM to develop Watson in the first place. Its natural-language interpretation ability and ability to sift through large data has allowed it to make connections between massive numbers of patient records that would be impossible for a human to process, and its intelligent advice to doctors has the potential to both reduce diagnostic errors and simultaneously improve the knowledge base of its human counterparts (Ferrucci et al. 2013). This same concept of computing power being used to reduce human error could also be applied in the military planning field.

Another system that many Americans are familiar with is TurboTax®, the tax software that guides its users through the process of filing their returns. Its goal is to streamline the 70,000+ pages of the U.S. tax code by only presenting the relevant portions to the user. It pulls previously saved data and inputs from external sources and compares them against the user's prompted inputs in order to find discrepancies or errors, using computing power to greatly reduce the administrative burden in filing tax returns (Goolsbee 2002). The smoothness of this experience for the user is one of the inspirations behind the development of TOOTH, and is one of the ultimate goals for the system.

D. FUTURE TECHNOLOGY

Understanding the existence of these technologies allows us to then examine what their merger might look like, both in terms of fitting in with the Marine Corps' vision for mobile computing and in terms of an ideal application for solving the identified technology gap.

1. Mobile Devices in the Marine Corps

The Marine Corps updated its Concept of Command and Control in September of 2015, with a particular emphasis on future combat and the systems necessary to operate in a network-degraded environment. Of note, the document continued to stress an emphasis on mission-type orders as the building blocks of combat communication:

> To achieve the adaptability and discrimination required in future combat, commanders must embrace decentralized command with mission-type orders, backed by a networked control system that informs the commander in a timely manner. Such a decentralized approach, coupled with shared understanding and applied in an environment of trust, enables subordinates to develop the situation, seize the initiative, create and exploit opportunities and cope with uncertainty. Use of mission-type orders will remain especially important given that many future enemies will attempt to disrupt U.S. and coalition information systems through computer network attack and other means. Commanders and their staffs must prepare and train for operations in an electronic denied or degraded environment, operating with locally available data and networks until connectivity can be restored. (6)

In a 2015 thesis examining mobile device procurement needs within the Marine Corps, Jesse Adkison highlighted the Corps' commitment to "providing an agile method to access information needs through the use of mobile devices," and specified ways that the service can utilize the Department of Defense's mobile device implementation plan to ensure joint application interoperability (Adkison 2015, 30). There is clearly an imperative to ensure that future technologies developed for the Marine Corps are done with an eye toward mobility and deploy-ability.

2. The Ideal Convergence of Theory and Technology

> A single program that manages all inputs and propagates any changes to all applicable documents (i.e, "TurboTax®-like approach for amphibious planning") would make the process more responsive and agile as well as reduce the probability of error. Existing automated tools, such as Joint Mission Planning System–Expeditionary (JMPS-E), were not used to their full potential due to the lack of trained personnel.

(Expeditionary Warfare Collaborative Team 2012, 6)

Rather than rely on potentially stressed or tired leaders to accurately chase their own changes around an orders format, we propose a paradigm that treats the operations order as one concrete and interconnected set of information points that can be manipulated, displayed, or exported in a variety of ways. To change a piece of information once and have that change replicated across an entire order and resulting products would allow leaders to focus more on their plans and less on the administrative burdens of communicating those plans in the standard written format. This is a fundamental disagreement with the recommendation found in the 2015 examination of Marine Corps mobile application needs, specifically that:

> The orders process may be better conveyed through other means than a mobile application. The warning and fragmentary orders are typically conveyed through verbal communications. The operations order is generally too lengthy to be delivered by any means other than textual. The order delivery process should not be considered for specific application development that is not already addressed with existing organic mobile device applications. (Adkison 2015, 68)

In order to be an effective tool at helping leaders create operations orders effectively and efficiently, an ideal software application would have the following characteristics:

1. *Mobile*. The software would run on a touch-enabled tablet device small enough to be easily portable on the battlefield, and would be able to function without a network connection, to include the ability to store maps and templates in local memory.

2. *Valid*. The software accounts for all of the different types of information present in the order, and makes links between the individual data points in ways that can be manipulated logically.
3. *Usable*. The user interface is designed in a way that is intuitive and easily manipulated by persons otherwise encumbered by fatigue, stress, or physical constraint.
4. *Focusable*. The data comprising an order can be used as the basis of a subordinate order or as an input to a higher order, thus saving the work of copying data between orders.
5. *Exportable*. The order data can be displayed in a variety of formats, ranging from the traditional ordered text output to various graphics or simulator-language formats.
6. *Assistive*. The application can guide the user in the creation of their order by suggesting next steps or by highlighting likely errors.

The full and detailed goal hierarchy established for the design and implementation of the prototype software can be found in Appendix A. Goal Hierarchy List.

E. SUMMARY

This chapter reviewed the Five Paragraph Order format in depth, and has identified both its reasons for existence and its inefficiencies—the numerous points at which the same information is used in different contexts. It is not the intention of this thesis to question whether or not the Five Paragraph Order itself should be redesigned. Instead, we embrace the utility that comes from having a universal format, and while we appreciate the need for information to be formatted in specific ways for subsets of the greater audience, we feel that manually replicating information multiple times in different ways throughout the order is an unnecessary waste of effort given the mobile computing technology available, let alone fraught with human error. The application design is specified in the following chapter.

III. APPLICATION DESIGN

The traditional method of software development is called the "waterfall" method, and it progresses in linear fashion through a series of steps: defining requirements, designing all interactions within the system, implementing the design, and finally testing the software. This method is primarily used for building well-defined programs to solve well-understood problems with a high degree of code reliability, but the initial steps can also be useful for organizing thoughts and determining overall system design before transitioning into an "iterative" or incremental approach to building software. The iterative method is much more responsive and useful for building nascent technologies: create a small portion of the application and then test that portion and its interactions with the rest of the system in a manner that can rapidly adjust to feedback and changing requirements (Braude 2001). We used the above combination of requirements definition with iterative development in creating TOOTH.

A. REQUIREMENTS AND GOALS

All software engineering projects should start with an examination of the requirements that need to be met, regardless of the development model chosen for the project. In his book *Software Engineering: An Object-Oriented Perspective,* Eric Braude says of requirements definition that "the great challenge we face is expressing clearly what customers want and need." He recommends starting with an examination of the "concept of operations" for the application before looking deeper at specific "use cases" (Braude 2001, 145). We follow the same approach here. Once we have defined the broad requirements, we outline them further here in a detailed goal hierarchy that can be used to design the data structures and user interfaces.

1. Concept of Operations

As referenced in Chapter I, the overall purpose of TOOTH is to be a one-stop shop for the creation and dissemination of operations orders. This means that the user should be able to create a complete, tactically sound, and communicable plan using only the application and its access to the host device's hardware and software tools (GPS, storage, radio card, input devices, etc.). It also implies that the application needs to be capable of addressing the complexities of the operations order process that were defined in Chapter II. Specifically, the software needs to be capable of pulling information from external sources, creating and manipulating information objects, performing operations on those objects, and then displaying or exporting those objects in the format chosen by the user.

2. Use Cases

Using Braude's method, the next step in detailing the application's requirements was to outline the specific tasks encompassed in the concept of operations. Each of these use cases may have multiple sub-steps, some of which are shared between use cases. The use cases are intended to address the eight characteristics of an ideal application that were introduced at the end of Chapter II (noted here in parenthesis).

1. Generate a new operations order: the user wishes to create a new order from scratch. This requires establishing a blank copy of the data structure to contain the user's inputs in a way that is uniquely distinguishable from other operations orders saved or edited on the device. (usable, assistive)
2. Save an operations order for future use: this case involves the ability to save the information in the operations order to the device's storage in a way that can be uniquely accessed at a later time. (usable, exportable)
3. Edit an existing order: this case presents the user with a list of operations orders that are saved on the device and reads the appropriate operations order from the saved file when selected by the user. (usable, assistive)
4. Create a new order from an existing order: this case presents the user with a list of operations orders saved to the device, opens the selected operations order, and then presents the user with a list of units saved within that operations order. The user selects a unit, and the application re-focuses the data structure to make the

selected unit the base unit for the new operations order. This use case shares steps with the previous use case, as demonstrated in Figure 13. (usable, focusable)

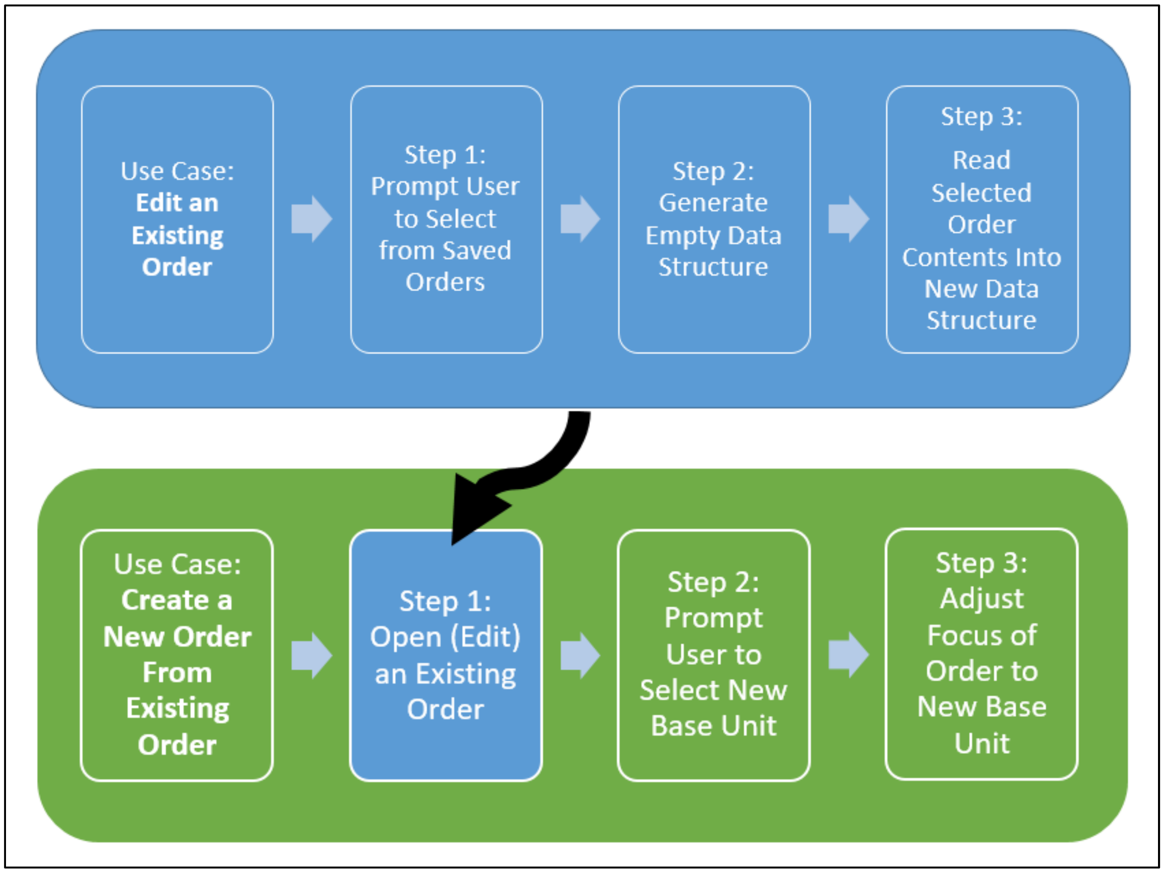

Use Case: Edit an Existing Order
Step 1: Prompt User to Select from Saved Orders
Step 2: Generate Empty Data Structure
Step 3: Read Selected Order Contents Into New Data Structure
Use Case: Create a New Order From Existing Order
Step 1: Open (Edit) an Existing Order
Step 2: Prompt User to Select New Base Unit
Step 3: Adjust Focus of Order to New Base Unit

Figure 13. Example of Shared Sub-Steps in Use Cases.

5. Guide the user through creation of an operations order: this case generates a new operations order data structure and then prompts the user to select the type of operation that they are trying to plan from a list of templates. The application then guides the user through the creation of their plan using lists of information objects that are doctrinally associated with the chosen operation type. (valid, usable, assistive)

6. Validate the current operations order: in this use-case the application applies a series of logical tests to the selected operations order in order to find omissions or inconsistencies. The logical rules are based on doctrinal templates and best practices, and they can be expanded throughout the development life cycle of

the application. This case can be initiated at any time during the user's workflow. (valid, assistive)

7. Export the operations order information: this case allows the user to package the data in a saved operations order into a variety of templates for export. Possible templates include:

- A Hypertext Markup Language (HTML) document that follows the traditional Five Paragraph Order format, compatible with any standards compliant web browser or word processing software.
- Map overlay files containing all of the Operational Terms and Graphics contained in the order. This includes unit locations, tactical control measures, and task graphics. File formats might include those necessary for viewing in GoogleEarth, C2PC, FalconView, Blue Force Tracker, and others.
- Documents needed by combat and support agencies assigned to provide fires or maneuver support to the operation. This includes target list worksheets, helicopter manifests, synchronization timelines, execution checklists, communications smart packs, and a host of other necessary de-confliction measures.
- Simulator definition files. This would re-package the information in the operations order into a battlefield management language or similar construct and allow leaders to test portions of their plan in a simulator or conduct large scale live/virtual training. (exportable)

8. Automatically generate mission-preparation and execution documents: this case examines the tasks and TCMs used in the order and generates mission preparation tools. Possible examples include:

   - Prioritized rehearsal plans focused on practicing the most essential tasks first.
   - Custom gear and equipment checklists tied to the tasks being conducted.
   - Vehicle loading manifests and "bump plans." (valid, usable, assistive)

3. Goal Hierarchy

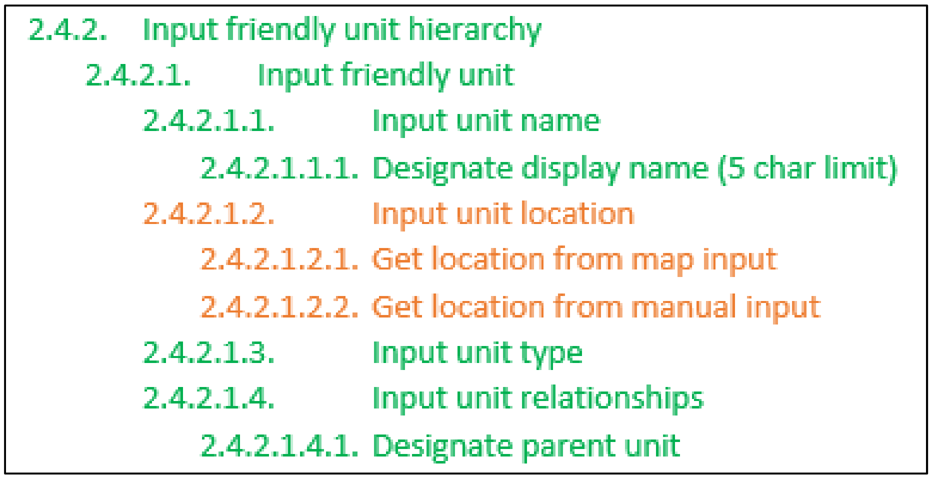
Translating these use cases into a series of programming tasks required the use of a goal hierarchy, a tool that traces its origins to the Work Breakdown

Structure (WBS) of project management originally made popular in the 1980s. Using a WBS approach can greatly help in defining the scope of a software engineering effort (Hans 2013). The goal hierarchy itself consists of a series of nested tasks that need to be accomplished in order for the software to perform as desired. The purpose of the hierarchy is to ensure that there are no extraneous tasks (those which do not support a higher level in the hierarchy), and that there is a roadmap leading to the successful completion of each high-level goal (Tausworthe 1979).

In building the goal hierarchy for TOOTH, we started with the use cases outlined in the preceding section and continued to add sub-goals until we reached a state where each goal could be represented by an individual function within the application software. Each individual goal is listed in the hierarchy once, and is referenced from other locations if necessary.

One example of a goal hierarchy is the use case for generating a new operations order with user input. Sub goals include initiating the data structure, designating the type of operation to be conducted, inputting the battlespace framework information, designating a unit mission statement, and inputting the scheme of maneuver and other major information categories in the operations order.

If we examine the sub-goal of inputting the battlespace framework information we find that it is comprised of entering known tactical control measures, along with friendly and enemy unit hierarchies. Within the goal of entering a friendly unit hierarchy, we find a sub-goal for inputting the information for a single unit, a goal that will likely require an interface with the user. This goal has its own sub-goals, including entering the units name, entering the unit’s location, its type, etc. We continued this examination until we had modelled the process of generating an operations order as it is taught to new lieutenants at TBS (The Basic School 2015a). This small example portion of the overall goal hierarchy is displayed in Figure 14.



2.4.2. Input friendly unit hierarchy
- 2.4.2.1. Input friendly unit
  - 2.4.2.1.1. Input unit name
    - 2.4.2.1.1.1. Designate display name (5 char limit)
  - 2.4.2.1.2. Input unit location
    - 2.4.2.1.2.1. Get location from map input
    - 2.4.2.1.2.2. Get location from manual input
  - 2.4.2.1.3. Input unit type
  - 2.4.2.1.4. Input unit relationships
    - 2.4.2.1.4.1. Designate parent unit

Figure 14. Example Goal Hierarchy Chain.

The goal hierarchy was a living document throughout the development of TOOTH, and was color-coded for quick reference to the progression of task completion. The full goal hierarchy is included as Appendix A.

4. Design Constraints

Before we transitioned from the goal hierarchy to the design that would accomplish those goals, it was necessary to ensure that the vision and the design principles applied would fit within the constraints imposed on TOOTH by both the Marine Corps and its user base. These constraints ultimately shaped both the design and implementation of the resulting technology demonstrator.

a. USMC Requirements

In order for TOOTH to ultimately achieve adoption within the greater Marine Corps community, it will be required to conform to the Defense Information Systems Agency's (DISA) requirements for commercial mobile device applications and its requirements for secure storage of classified information. Because encryption and secure communications are outside of the scope of this thesis, that compliance effort is left for future work. Although TOOTH in its current form as a technology demonstrator will not contain any classified information, efforts were made during design and implementation to ensure that the data structure used will support best practices for the security of data at rest on the device.

b. Device Constraints

TOOTH is designed for adoption into the Marine Corps using existing gear and equipment. This means that it needs to run on the Android tablets that the Marine Corps has already fielded for running KILSWITCH software as part of the Precision Close Air Support System. Those tablets use versions of the Android operating system as old as API version 17 (NAVAIR 2015). Furthermore, the application needs to be designed for use on the screen sizes most likely to be carried by Marines in expeditionary environments. This means designing for a mobile device that fits in a user's cargo pocket with a diagonal screen size of 8 inches or below.

c. User Constraints

Additional constraints are imposed by the capabilities and operational limitations of the target users of TOOTH, which must be navigable by the "least common denominator" within the user base. This user is basically trained, but is not very familiar with either the steps or the desired outcome of the orders generation process. We also expect that while this user is familiar with mobile technology, he or she will most likely not be comfortable navigating the device's file structure in order to make changes or find files. The application must compensate for those deficiencies, as well as others that come as a result of the user's expected operating environment:

- Fatigue – We expect the user will be suffering from the general effects of sleep loss and will have difficulty concentrating on the task at hand or remembering simple steps in the orders generation process.
- Diminished vision – We expect the user to have some difficulty viewing the screen of the device, whether due to using the tablet in low-power mode (such as for light-discipline reasons at night or to save battery-life) or due to straining to see the screen in bright sunlight.

- Diminished fine motor control – We expect the user to have some difficulty making fine muscle movements. This could be due to muscle fatigue, anxiety, parasympathetic backlash (Grossman and Christensen 2007) or other factors, such as having to operate the device while wearing gloves.

These constraints drove several key areas in the design of the application logic and the user interface.

B. OVERARCHING DESIGN

Because the goal of TOOTH is to provide a one-stop-shop for the entire orders process, it is essential that the user be able to access all of the tools that they need without leaving the application. TOOTH will save and load files from the device storage and prompt the user to turn on GPS. The application will also leverage the operating system's soft (on-screen) keyboard and touchscreen display to provide an input method to the user.

The internal workings of the application will also seek to simplify the process for the user. There will only be one order loaded in the application at a time, with a backup order slot being used to save / auto-save and switch orders. The system will automatically present the order information from the designated base unit's perspective, and will only allow the user to change perspectives by generating a new order from the base order. Only one order can be active at a time, so there is no confusion over which set of data is being examined in the user check and guide functions. The high level software-hardware interactions are displayed in Figure 15.

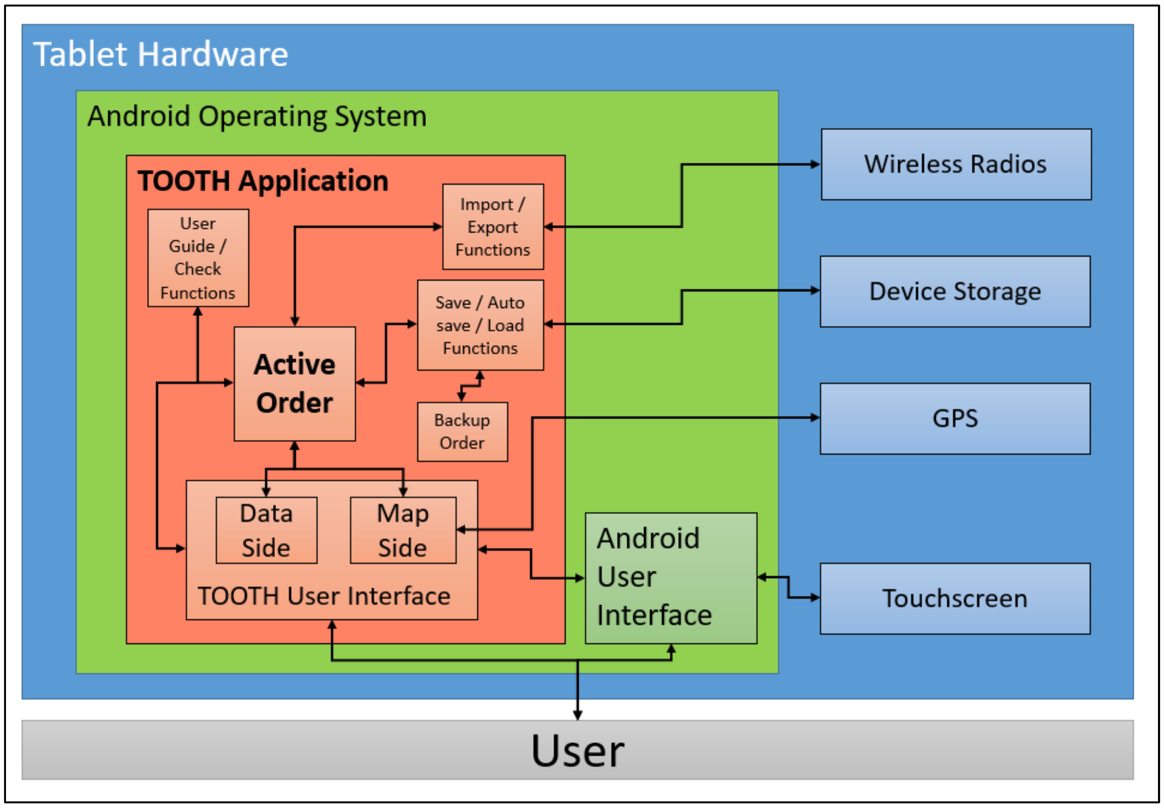


Figure 15. Overview of System-Level Design and Interactions.

C. DATA STRUCTURES

The central idea of this thesis is the assertion that viewing the information in an operations order as a set of inter-connected objects provides a framework for achieving significant increases in efficiency and accuracy. Thus, the design of the data structure is critically important to the success of TOOTH. Another key idea from Chapter II is the principle that the software needs to be "focusable" in that it can generate a new order shell from a higher, subordinate or adjacent unit order. For this to be implemented, the data structure needs to be extensible in all directions. A battalion commander should be able to issue an order to company commanders, who in turn issue it to platoon commanders, who in turn issue it to squad leaders, etc. The original battalion order information is retained all the way down the chain of command, with new layers and objects being added to the structure at each level. This basic idea is demonstrated in Figure 16.

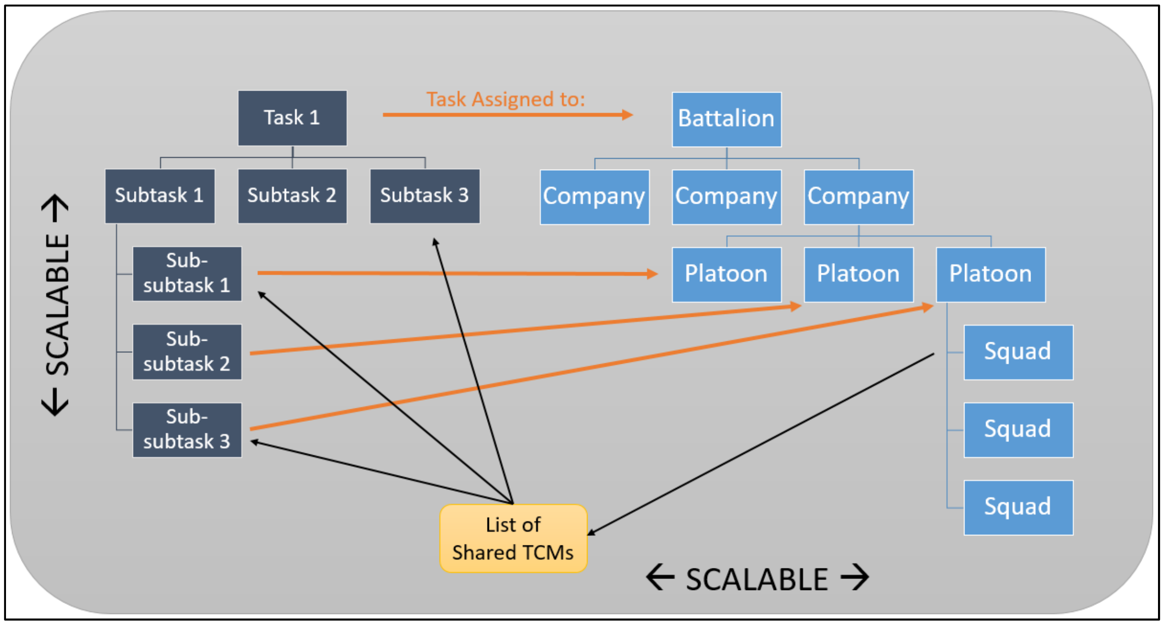


Figure 16. Example Diagram of Data Scalability.

1. Using the Object-Oriented Concept

Another way that the data needs to be scalable is in the quantity and types of each piece of information. An operations order at the battalion level may contain information on dozens of units, scores of tasks, and hundreds of tactical control measures. Each of these information objects will share some common features, but may be unique in others. The system is perfect for modelling with an object-oriented approach, with each object representing a specific class of information.

C. Thomas Wu's *An Introduction to Object-Oriented Programming with Java* (2006) defines it this way: "In object-oriented programming, we use a mechanism called inheritance to design two or more entities that are different but share many common features. First we define a class that contains the common features of the entities. Then we define classes as an extension of the common class inheriting everything from the common class" (23).

For the TOOTH data structure, we started with a root object definition called OrderDataObject from which all other data types are extended. This root object has parameters for information such as the date and time that the object is created or modified and the application user who created or modified it. It also contains a string field that is set according to the object's specific type. An example of a specific type of order data is the case of tactical control measures (TCMs)—named points, lines and areas on the map that serve as reference points for tasks and other coordinating instructions. In this application, all TCMs share certain attributes. They all have a name and an associated faction (friendly, enemy, etc.) in addition to the attributes that they inherit from the base OrderDataObject class. Within the TCM class are several sub-classes based on the unique types of TCMs. Some are defined by a single point on the map, some are lines, some are circles, and others are two or three dimensional areas. Within each of those types are further sub-classes. Checkpoints and point targets are both examples of point TCMs, for example, but they are used in very different ways by the other information in the operations order. This tree of relationships is depicted in Figure 17.

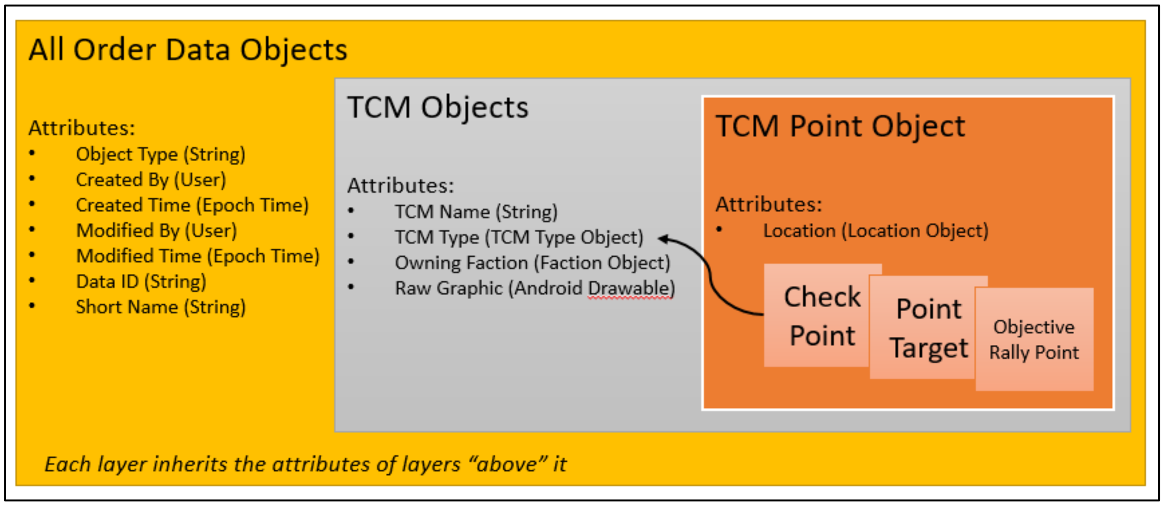


Figure 17. Example of Object Inheritance in the TCM Class.

Another benefit of using an object-oriented approach is that objects can have their own "methods," or functions. In the context of TOOTH, for example, a TCM object can have a method that adds its information to a master list of TCMs upon its creation, or removes it from the order data structure if the user calls for its deletion. This ability for objects to perform operations on themselves and the data stored in them is a large part of the application's proposed user guiding and data validation functions.

2. Defining Data Interactions

The next step after building the full inheritance tree containing the various information objects that comprise the operations order was to define the links between those objects. Besides inheritance, the other basic object relationship is association—linking two objects together by having one (or both) of the objects contain a parameter that links to the other. Association links can be one-to-one, one-to-many, or many-to-many (Braude 2001). For a TOOTH example, every TCM objects contains (links to) one or more geographic locations, each of which is a unique data object containing latitude, longitude and altitude data.

In order to determine the full set of linkages between information objects, we retraced the operations planning process taught at EWS (United States Marine Corps 2010) and the tactical order generation process taught to new officers at TBS (The Basic School 2015a) and cross-referenced those processes with the official definitions of operations and terms found in Joint doctrine (U.S. Joint Forces Command 2011; Department of the Army 1997) and the official definitions of operational terms (Department of the Army 2004) in order to ensure that the data interactions were modelled in a doctrinally sound manner. A portion of the full graph is shown in Figure 18.

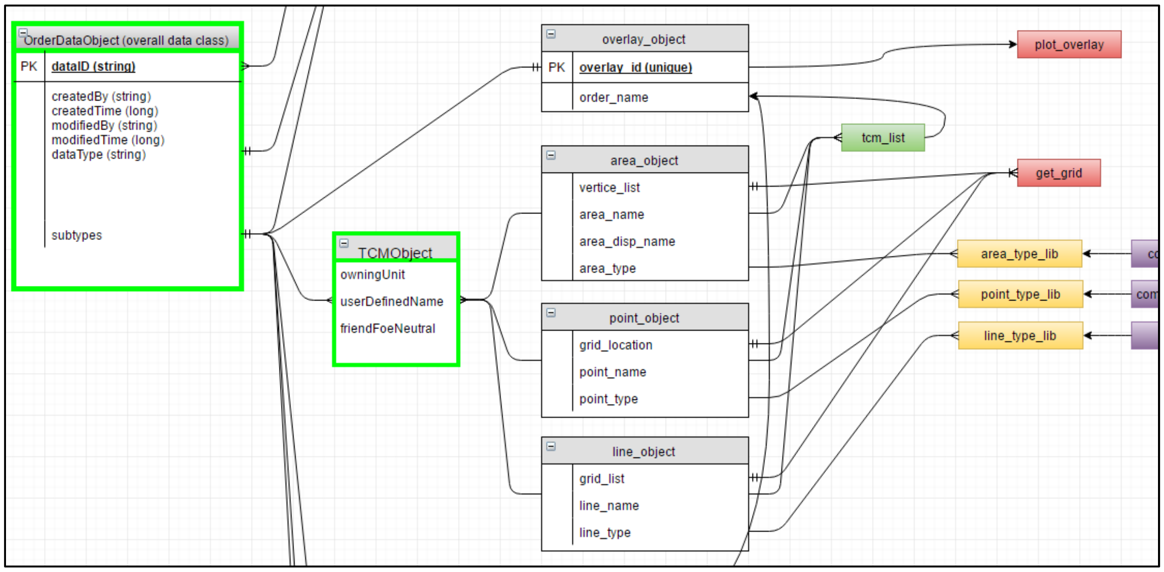


Figure 18. Example Graph of TCM Data Object Interactions.

The relationships between information objects not only need to be defined, they also need to be reinforced. There are a number of one-to-one and one-to-many relationships defined in the data structure. If a link is made between two objects and one of those objects is later deleted, the link needs to be deleted as well. Otherwise, one of two things is likely to happen: the application will crash because it attempts to follow a link to a non-existing object (a “null pointer” error), or the order information presented to the user will be wrong because the data structure will have failed to maintain consistency (this is the more insidious error). The latter defeats the entire promise of a system in which changes made to one part of the operations order are automatically replicated to others. Figure 19 demonstrates an example of the logic used to ensure data consistency.

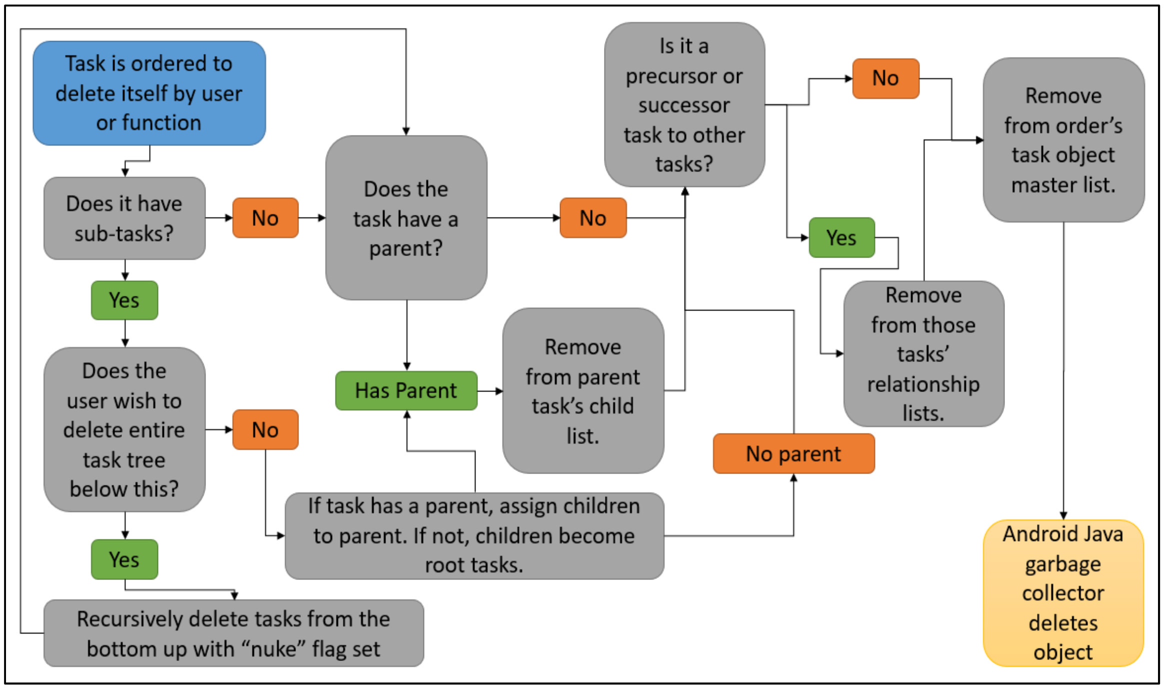

Task is ordered to delete itself by user or function
Does it have sub-tasks?
No
Yes
Does the task have a parent?
No
Has Parent
Is it a precursor or successor task to other tasks?
No
Yes
Remove from order's task object master list.
Remove from those tasks' relationship lists.
Remove from parent task's child list.
Does the user wish to delete entire task tree below this?
No
Yes
No parent
If task has a parent, assign children to parent. If not, children become root tasks.
Recursively delete tasks from the bottom up with "nuke" flag set
Android Java garbage collector deletes object

Figure 19. Example Task Object Method Logic for Consistency in Self-Deletion.

3. Data vs. Format Focus

The resulting graph of the interactions between all of the information objects showed a contrast between the Five Paragraph Order Format and the underlying data. The graph especially highlighted the importance of tasks, since both Paragraph 2 (Mission) and the scheme of maneuver are essentially re-wording of tasks. Similarly, the friendly and enemy unit hierarchies prove central to making sense of both the tasks and the tactical control measures. The data structure also highlights the areas that will be most important in designing the data validation function of the application—the more heavily-weighted a data object's connections to other objects the more important its role in the order as a whole. The full data relationship graph is included as Appendix B.

D. USER INTERFACE

In his book, *Android User Interface Development*, Jason Morris (2011) defines the "core guidelines which every user interface should follow":

- Consistency – Items in the user interface should move as little as possible, so as to be predictable to a user. Furthermore, the user should be able to identify which items in the interface are buttons, and should be able to anticipate the results of a given action.
- Simplicity – The user interface should allow the user to accomplish their intended task in the fewest number of steps possible, while simultaneously limiting their options to keep from overwhelming them. This is especially important for new users.
- Feedback – The interface should respond to a user's actions and should provide a navigation mechanism that helps the user know where they are in the interface at any given time. (21).

1. Overarching Goals

Although the central idea of this thesis is tied to the functioning of the data structure as a way of interconnecting all of the information in an operations order, that idea will be lost on the target users if it is not implemented in a way that is accessible to the average user. This requires a user interface that presents the information in a way that is not only tailored to the characteristics of the data being displayed but is also both useful and familiar to the user.

Military operations are inherently organized and described using time and space, so the user interface must have a way of graphing the information on a map while also displaying elements such as tasks in relations to each other on a timeline. The initial design sketches attempted to merge those two ideas, either through a map with a series of pop-up windows that could be accessed from the edges or on a map with a timeline "slider" that would allow the user to see the progression of their plan by changing time as a dependent variable. These initial sketches are shown in Figure 20 and Figure 21.

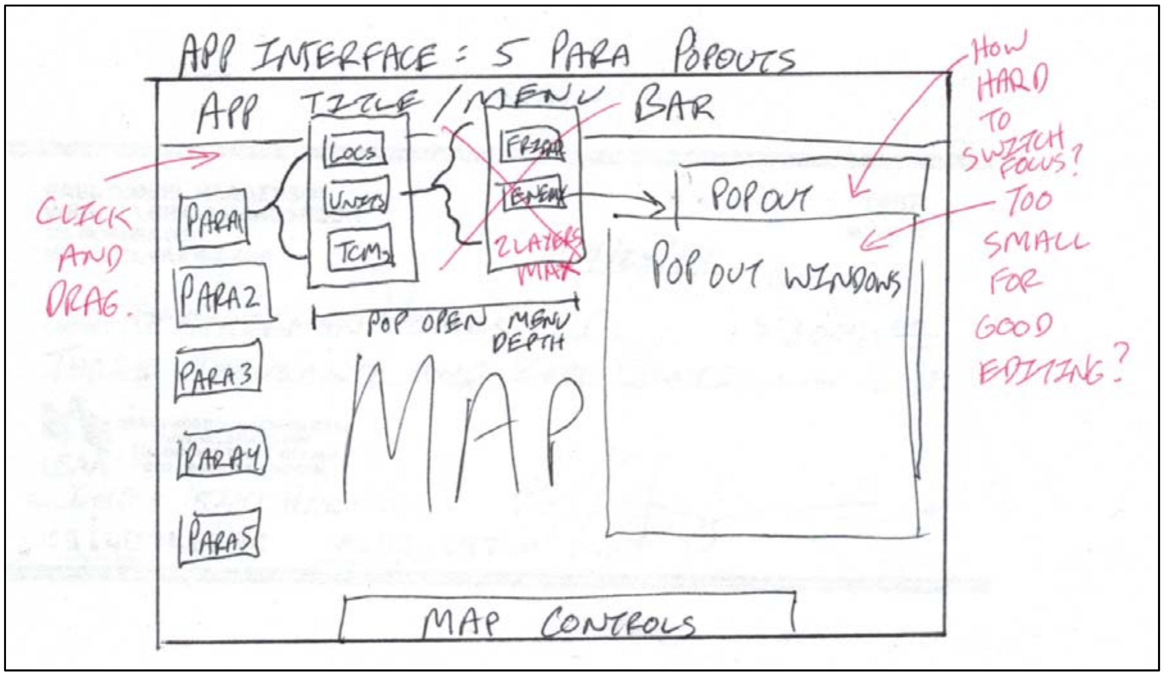


Figure 20. Initial User Interface Design Idea – Pop-up Windows Overlaid on Map.

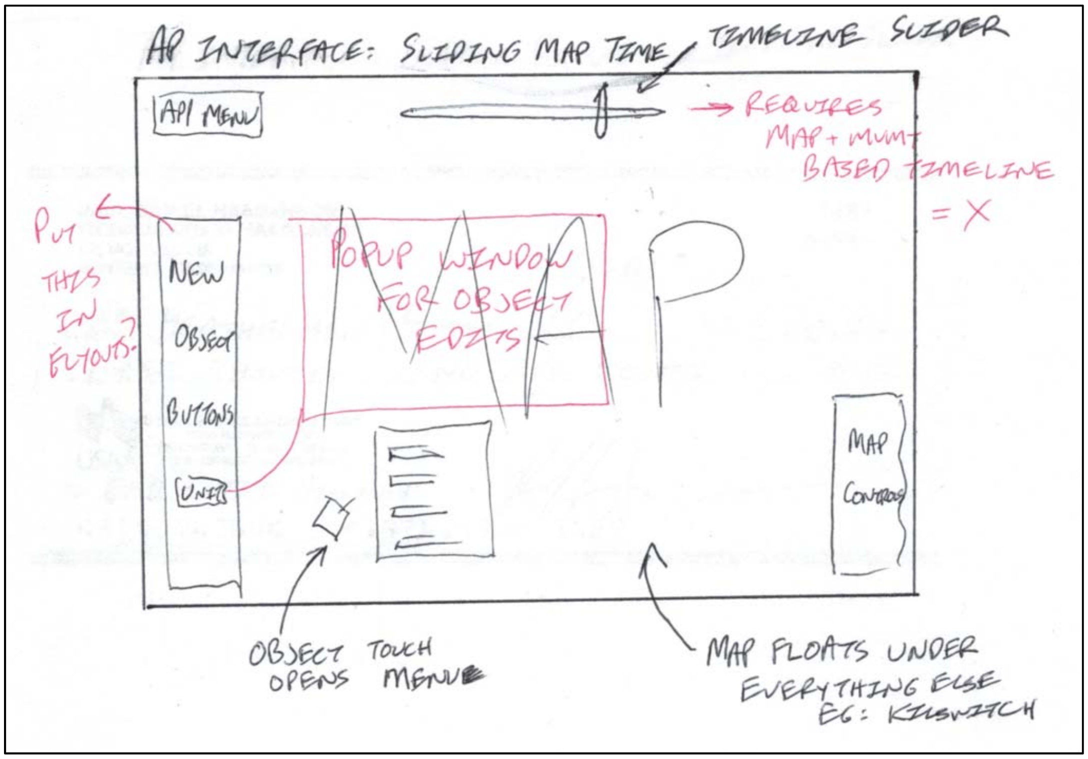

AP INTERFACE: SLIDING MAP TIME
TIMELINE SLIDER
APP MENU
REQUIRES MAP + MUMT BASED TIMELINE = X
PUT THIS IN FLYOUTS?
NEW
OBJECT
BUTTONS
UNITS
POPUP WINDOW FOR OBJECT EDITS
MAP
MAP CONTROLS
OBJECT TOUCH OPENS MENU
MAP FLOATS UNDER EVERYTHING ELSE EG: KILLSWITCH

Figure 21. Initial User Interface Design Idea – Map-as-a-Timeline.

We felt that the map-centric interface with the array of buttons around the edge was un-necessarily cluttered and had the potential to be confusing—we felt the user would have a hard time knowing where they were in the application's logic. Similarly, the idea of changing the map based on a timeline slider seemed novel, but it might also be extremely counter-intuitive and discouraging to a novice user. It would be more difficult to program and would probably require more fine-motor actions on the part of a fatigued user. The end result was a compromise (Figure 22) that placed a thin title bar and application-level menu above an almost full-screen window that could be "flipped" between two different views: an interactive map as a logical way to display all plot-able data points and an organized display of compartmentalized windows structured along the familiar Five Paragraph Order Format. The user would flip back and forth between the two interfaces with a button and, because both interfaces access the same underlying data structure, changes made in one "side" of the application would automatically be displayed on the other, and vice versa.

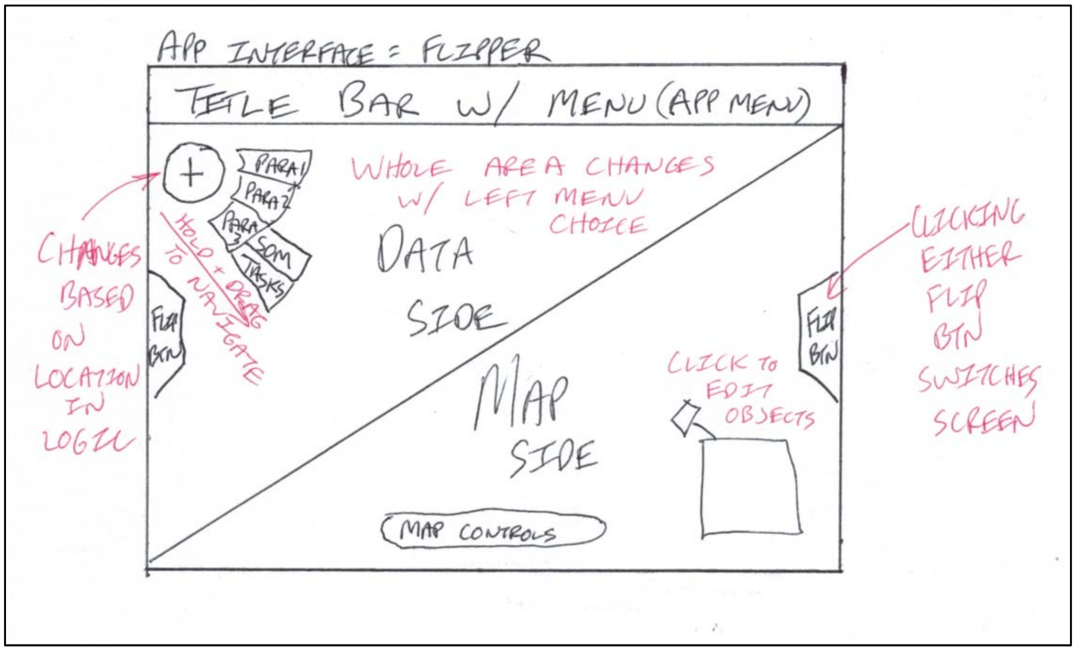
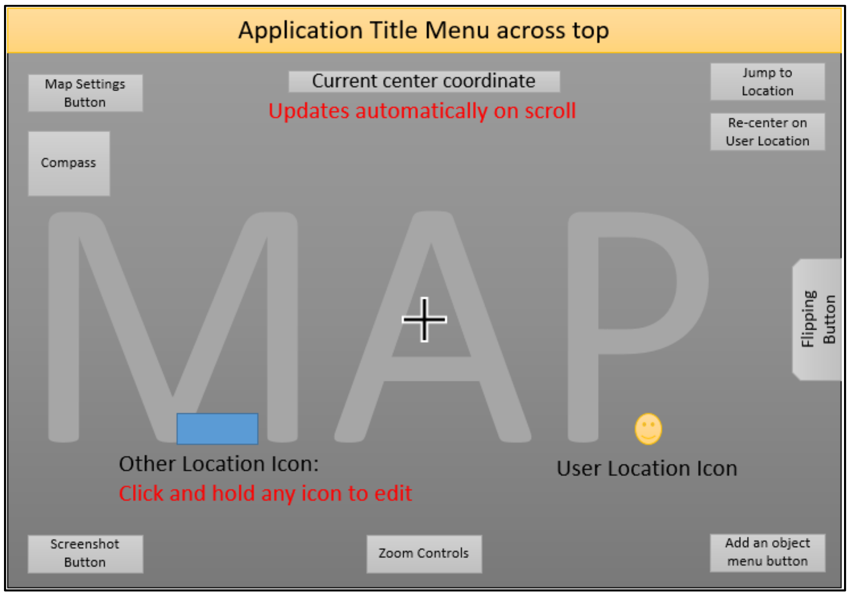

APP INTERFACE = FLIPPER
TITLE BAR W/ MENU (APP MENU)
+
PARA 1
PARA 2
PARA 3
SOM
TASKS
HOLD + DRAG TO NAVIGATE
WHOLE AREA CHANGES W/ LEFT MENU CHOICE
DATA SIDE
CHANGES BASED ON LOCATION IN LOGIC
FLIP BTN
FLIP BTN
CLICKING EITHER FLIP BTN SWITCHES SCREEN
MAP SIDE
CLICK TO EDIT OBJECTS
MAP CONTROLS

Figure 22. Initial User Interface Design Idea – Compromise Model with Radial Menu Button.

Most elements of this compromise design were incorporated into the technology demonstration software, although the menu design proved to be unwieldy and would have been difficult for a user to navigate with shaky or gloved fingers.

2. Map Display Design

The goal of the mapping portion of the software was to provide the user with a large map canvas that operated intuitively according to the motions she is used to using on her personal mobile device. The map should be zoom-able and scrollable, and it should allow the user to get the coordinates of a point simply by clicking on it. The map should also integrate with the device's GPS application to position a user location icon that is constantly updated. Users should be able to create and edit any information object with an associated location parameter simply by clicking on the map or selecting options from a short menu. The map should also implement an on-screen compass and should allow for both online and offline map storage. In the end, the design template looked very similar to other mapping software due to the overlap in required capabilities (Figure 23).



Application Title Menu across top
Map Settings Button
Current center coordinate
Updates automatically on scroll
Jump to Location
Re-center on User Location
Compass
MAP
Flipping Button
Other Location Icon:
Click and hold any icon to edit
User Location Icon
Screenshot Button
Zoom Controls
Add an object menu button

Figure 23. Map Window Template before Implementation.

3. Data Entry Design

The data entry part of the user interface was harder to design than the map. Unlike the map, which simply graphs all location-based information objects in relation to each other, the data entry part needs to display different views depending on the type of information being collected or displayed.

Most data types can be added and updated through the use of simple forms, with text fields for entry of simple parameters such as names. Any associations between objects can be established by prompting the user to select objects from a drop-down list of eligible candidates. This method is seamless to the user, but as discussed in Chapter IV, it is not trivial to program. Other lists of items are similarly easy to conceptualize. For example, when the user is prompted to load a saved order, the device can simply query its storage and return a list of all saved files with the proper extension. Once the user selects an option from the list, the program performs the necessary operations to load the chosen order data into memory.

Lists are not the most appropriate way of displaying other data in the operations order. A list of units, for example, does a poor job of showing which units are subordinate to the others. For this reason, written orders usually depict unit relationships in a table of organization (Figure 24).

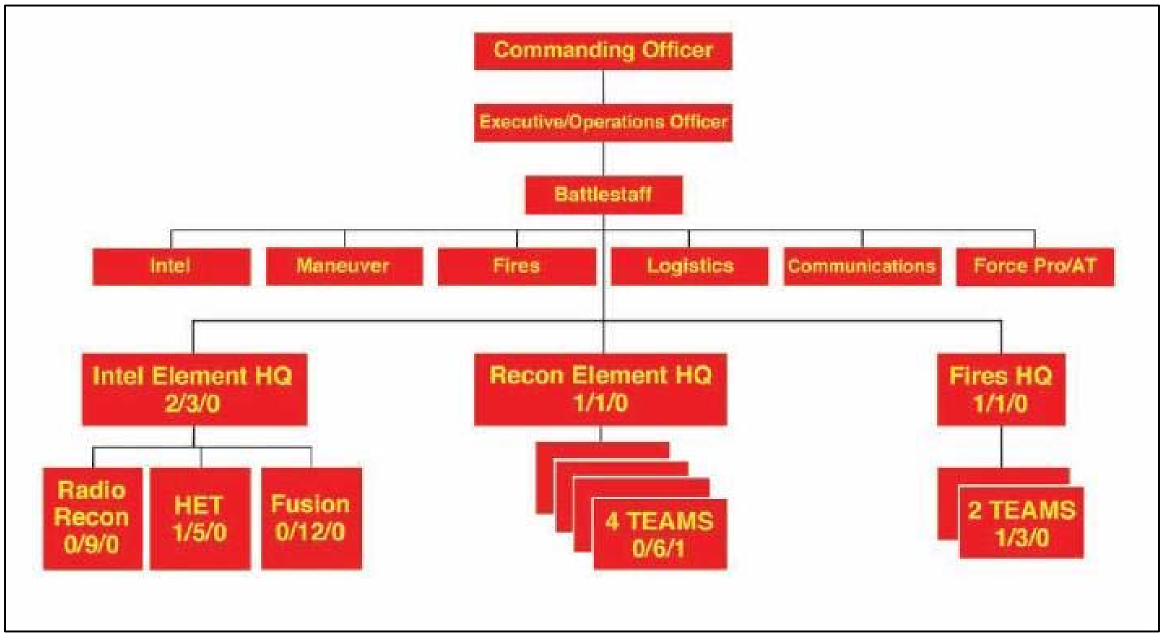
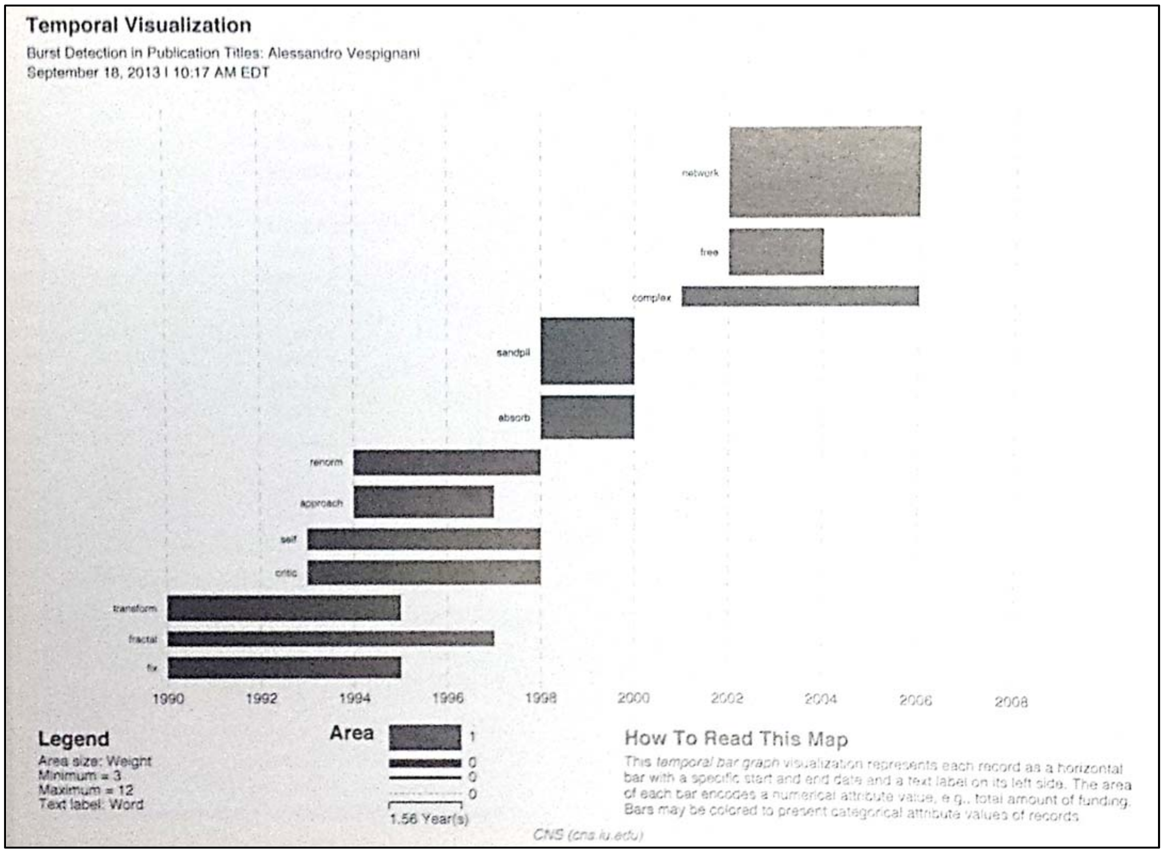


Figure 24. Example Table of Organization for a Marine Corps Special Operations Unit. Source: Global Security (2017).

This tree-like view provides an informative and useful way to visualize unit relationships. We decided that the unit portion of the user interface should provide a "clickable" unit tree that automatically updates with every change to the unit structure. This allows users to see their changes in real time, and also allows them to change other information elements associated with the units by selecting them in the user interface. For example, in Figure 24, clicking on "Fires HQ" should provide a menu of options for that unit such as "Change Unit Location" or "Modify Unit Information."

Similarly, a list of tasks does a poor job of showing their relationship to one another with respect to time. Instead, tasks are best displayed on an overlapping timeline similar to the Gantt chart discussed in Chapter II. One way of building this type of graph is the "Temporal Bar Graph" discussed in their book *Visual Insights: A Practical Guide to Making Sense of Data* by Katy Börner and David Polley (2014) (Figure 25).



This temporal bar graph visualization represents each record as a horizontal bar with a specific start and end date and a text label on its left side. The area of each bar encodes a numerical attribute value, e.g., total amount of funding. Bars may be colored to present categorical attribute values of records.

Figure 25. Example of a Temporal Bar Graph. Source: Börner and Polley (2014).

The ideal solution we envisioned for a task interface is a combination of a temporal bar graph and a Gantt chart, showing the overlap between different tasks while at the same time displaying the relationships between them in a way that indicates which tasks are pre-conditions of other tasks. Similar to the unit interface, clicking on a specific task in this interface should enable the user to edit its properties, define new task relationships, or edit its spatial characteristics with respect to the map. This interface also needs to update automatically upon user changes.

E. SUMMARY

This chapter detailed the organization and design work that was necessary in order to set conditions for the actual implementation of the technology demonstrator. It discussed the design ideas for a flexible and multi-layered object-oriented data structure and discussed ways to make that data structure intuitive to the user in the form of a custom user interface. Chapter IV discusses the work done to actually produce the TOOTH application, including the ways that these ideas were implemented.

IV. DESIGN IMPLEMENTATION AND TESTING

In order to effectively demonstrate the ideas and software design that were detailed in Chapters II and III, it was necessary to create a working application to serve as the technology demonstrator—software that could be manipulated by users on an actual physical tablet computer. We prioritized the goal hierarchy to give precedence to the functions that were essential to conveying the central idea of the thesis. That prioritization is reflected in the full goal hierarchy found in Appendix A.

A. IMPLEMENTATION PLAN

The author's original intent was to build the software in the same order that its design was developed—a complete implementation of the data structure followed by a user interface to manipulate the data. This quickly proved infeasible, as there was limited ability to test the functioning of the data structure without creating the means to interact with the individual data objects.

1. Agile Method

Instead, the author adopted a more "agile" approach to the development of TOOTH. The agile family of methodologies is highly iterative and embraces early requirements changes as an opportunity to take projects in useful directions. It also values creating working software more than the organization that comes from following a formal process. It is especially useful for creating new technologies where formal definitions and documentation do not exist (Highsmith 2004). For the development of TOOTH, this method became a rapid loop of creating functions and then testing them against existing elements of the software, fixing obviously broken elements of the technology demonstrator but bypassing formal proofs in favor of a physically demonstrable product.

2. Android Studio Programming

As discussed in Chapter II, TOOTH or any subsequent software iterations must be written for Google's Android operating system in order to be compatible with currently-fielded USMC devices. Android uses an implementation of the Java programming language for its programming logic and a series of Extensible Markup Language (XML) files to define the visible layout of elements on the screen (Morris 2011). Android Java is very well-documented, with an official Application Program Interface (API) available on Google's website for developers to make use of features available in the Android operating system (Google 2017). Google also provides a robust and open-source Integrated Development Environment (IDE) called Android Studio that simplifies a large amount of the mundane work of writing Android Java code. The final version of Android Studio used in the development of the TOOTH technology demonstrator was version 2.3.1.

3. Application Debugging

The primary method of error detection and correction used during development of TOOTH was the practice of "wrapping" potentially faulty code elements with Java try / catch statements that utilized custom error messages. This allowed the author to follow the flow of function calls in the IDE's operating system monitor and note which elements of the code were causing the application to throw error codes. This method, combined with the ability to "step through" the code execution by setting break points in the IDE, allowed the author to quickly determine faults in the code or recover system-generating error messages that could then be used as search terms in the software development forum, StackOverflow.

Although there are some elements of TOOTH that lend themselves to formal unit testing (such as the functions that automatically sort and display tactical unit relationships), the fact that most data operations in the software are tied to user interactions made it difficult to design effective unit tests. The author opted instead to design the software in such a way that it appropriately handled predictable input mistakes, and then used a trial-and-error approach to finding and correcting issues in the software logic. For example, the author tested combinations of completed and blank data entries on user-interface forms in order to ensure that the application did not attempt to create links to non-existent objects. Although this method was effective for finding and fixing errors that might have prevented a working technology demonstrator, it is by no means sufficient to ensure that the logic of the software is immune to crashing based on user inputs. A formal unit test of the data structure is part of the recommended future work discussed in Chapter V.

4. KILSWITCH Plugin Integration

The original vision for TOOTH was that its data structure and logic could be nested within the KILSWITCH application (discussed in paragraph II.C.1.d) in the form of a “plug-in”—an application that makes use of the KILSWITCH API to build functionality into the already-robust mapping software. This remains a possible future avenue for development of TOOTH; however, by the time the author had learned enough about Android application development to make use of the KILSWITCH API, the implementation of the TOOTH data structure and user interface had already progressed to the point that integrating with KILSWITCH would have required a time-intensive reworking of already-completed functionality. Furthermore, the KILSWITCH API only allows the developer partial control over the user interface and limited control over the application’s interactions with the Android operating system (NAVAIR 2016). Instead, the author made the decision to proceed with development of TOOTH as a stand-alone application utilizing an open-source mapping capability for the purpose of ensuring a working technology demonstrator.

B. APPLICATION STRUCTURE

The central building block of Android applications is the “activity” (Google 2017). Activities directly interface with the operating system, and are created, paused, restarted, stopped, and destroyed by the operating system based on the actions of the user. An application by default consists of a single activity and exists as a single process and thread within the Android Linux kernel (Google 2017). However, applications can consist of multiple activities, and developers can choose to open additional threads in order to run elements of the application asynchronously. These separately-threaded activities communicate with each other using messages called "intents" (Google 2017). The basic life cycle of an Android application activity is shown in Figure 26.

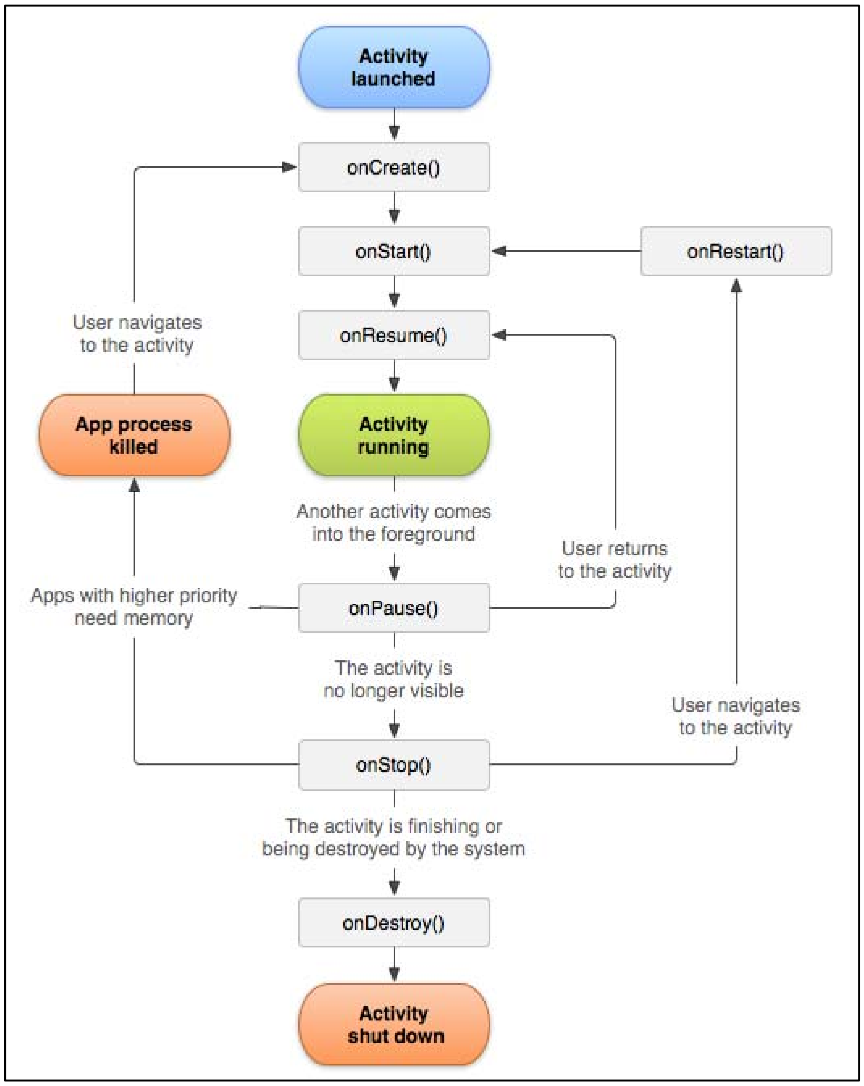

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 26. Life Cycle of an Android Application Activity. Source: Google (2017).

TOOTH is a single-activity application that conducts the vast majority of its operations on a single primary thread. Some minor elements of the application, such as the auto save and power management functions, run concurrently with the main application in their own separate threads. Additionally, elements such as the map function and the file saving function have the ability to open helper threads that run in the background in order to accomplish tasks such as downloading additional map data. Despite the potential for delay during times of intense processor usage, the decision to execute the data and user interface functions of the application on a single thread was a deliberate one. This decision streamlined the development of the application while simultaneously ensuring data integrity. The application logic executes linearly in a single thread, so there are no race conditions in which multiple threads may attempt to manipulate data at the same time. This eliminates the need for data locks and semaphores to ensure data consistency, and greatly simplifies the requirements for the application's interactions with its data. This decision will likely need to be revisited as the application grows in complexity with future work, but in this implementation the delay experienced by the user as a result of synchronous execution is negligible.

The primary activity in TOOTH is called "ToothMain" and is the first java class loaded upon initiation of the application. Figure 27 provides pseudo-code for the actions completed by ToothMain.

```
Algorithm Tooth Main Application
Input: None
Output: None
populate list of saved users
initiate data structure
establish global variables
if application loaded for first time:
    show disclaimer message

establish data content fragment
establish map content fragment

start power management, auto save and connection check threads

if current user == null:
    display user menu

else if content fragment is not visible:
    load main content fragment

start listeners for all menu buttons
```

Figure 27. Pseudocode for TOOTH Activity Initiation.

1. Data Object Structure

The data structure (OrderData java class) instance generated by TOOTH at application initiation is the container for all of the objects that comprise a given operations order. Its attributes include information such as the order title and description, the user who created or modified the order, the time the order was created or modified, and the unit designated as the base unit for the order. It also contains a separate list for each of the specific order data object types. Whenever an object is created or deleted, a reference to the object is added to (or removed from) the associated list. This allows for functions to iterate through all order objects of a specific type, and also prevents the Java garbage collector from deleting objects that are relevant to the operations order.

a. Object Serialization

One feature of the Android Java language is its ability to serialize objects. This essentially converts the object’s attributes to a byte stream that can then be written to storage or transmitted to another device or application. An exact replica of the object can be reconstructed as long as the target device/application has a copy of the object’s class definition file. This is required because the serialized byte stream does not carry the methods inherent in the object, only its attributes (Google 2017).

TOOTH implements serialization on all order data objects so that they can be saved to the device’s storage and reloaded into memory without having to be recreated by the user. This allows the order data to be saved between sessions or shared with other devices that have the TOOTH application installed. It also allows multiple copies of the same object to be compared attribute-by-attribute, which in turn allows for version control and enables functionality that would allow information from multiple subordinate operations orders to be absorbed into a higher-level operations order.

b. Default Object Generation

The central premise of TOOTH is that all of the information in an operations order can be represented as a set of interconnect objects. This requires a library of “descriptor” objects that provide additional attributes for object instances created by the user. For example, the tactical task object class has an attribute “task type” that allows a user to select the type of task they are creating from a doctrinally correct list of tasks. The user could select the descriptor object for the tactical task “block” in this example, and the application would use the attributes of the descriptor object called “block” to extract some facts about the tactical task object being instantiated: that it should be assigned to a friendly unit, oriented against an enemy unit, contain links to certain tactical control measures, and what its map icon looks like. Because there is only one doctrinally-correct definition for the “block” task, there only needs to be one "block" task-type object resident in memory. As many new tactical task instances as necessary can be linked to it.

This paradigm requires that the descriptor objects be already instantiated and available to the user before they attempt to create individual tactical task objects. To ensure that this is the case, each new OrderData object instantiates the libraries of descriptor objects at its creation using a series of internal methods. The pseudocode for one such method, titled "createDefaultTaskTypes," is given in Figure 28.

```
Algorithm Create Default Task Types
Input: None
Output: Task type objects initiated
default task types (array) = built-in XML resource file R.array.tasks
length (integer) = length of default task types
temporary array (string array) initiated

for i in range (length):
    add task type item attributes to the temporary array

for i in range (length):
    item long name = temporary array [i][0]
    item short name = temporary array [i][1]
    item description = temporary array [i][2]
    item target type = temporary array [i][3]
    temporary object = new task type object (long name, short name, desc., target type)

    add temporary item to order
```

Figure 28. Pseudocode for Creating Default Task Type Objects.

The function first looks in the device's storage for a saved list of task types. If a list of task type objects is not found on the device, the application uses a packaged XML library file to instantiate the default objects and save a new copy of the list to the device. Loading a saved list of serialized objects is the preferred method because it takes less time than regenerating the default objects from the packaged XML file. The inclusion of the regeneration method ensures that TOOTH can continue to function even if a user deletes all of their saved data from the device.

c. User-Defined Object Creation

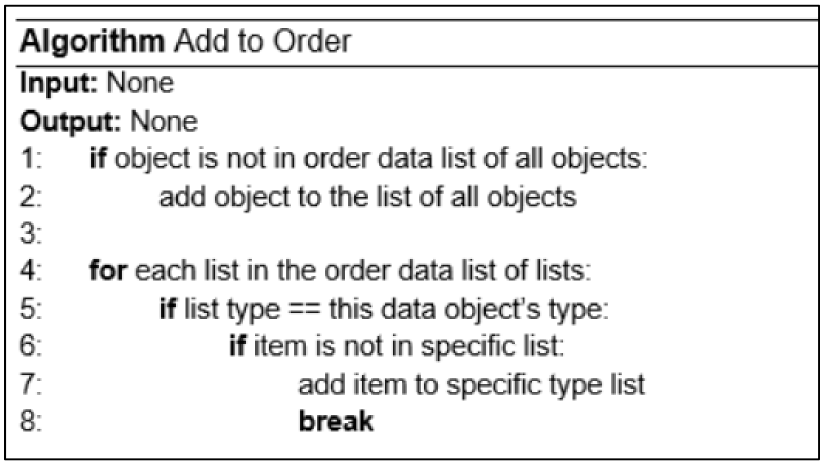
Another benefit of storing a list of serialized default objects is the ability to offer the user the option of creating their own default types. For example, there is no doctrinal tactical task for “engage local leaders in conversation” but it is very likely that a user planning an operation in a counterinsurgency (COIN) environment might want to plan a patrol around this type of task. A simple form interface allows the user to define the attributes of this type of task and save it to the list of doctrinal task types that are prepackaged with TOOTH. Once the user has finished entering the information, the application generates a new task type object with the specified attributes, adds it to the task type list, and then overwrites the stored task type file with the new list of objects. The process of storing and reloading objects is described in detail in Paragraph 2.a.

2. Data Manipulation

Another concept central to the goals of TOOTH is the idea that changes to information in one part of the order are instantly and automatically represented throughout the order. This happens because of the linked nature of the information objects. If the user changes a unit’s name, for example, that name change is reflected in the unit hierarchy and is also reflected on the map icons. If that unit had subordinate (or “child”) units, then clicking on a subordinate unit would reflect the name change of its parent. The manipulation of data attributes is possible because the data object classes contain methods called “setters” that enable modification of individual attributes. These setters can be used in conjunction with a user interface form to enable the user to quickly adjust attributes by means of the device’s standard inputs. The implementation of these user interfaces is described in detail in Paragraph C.3.a.

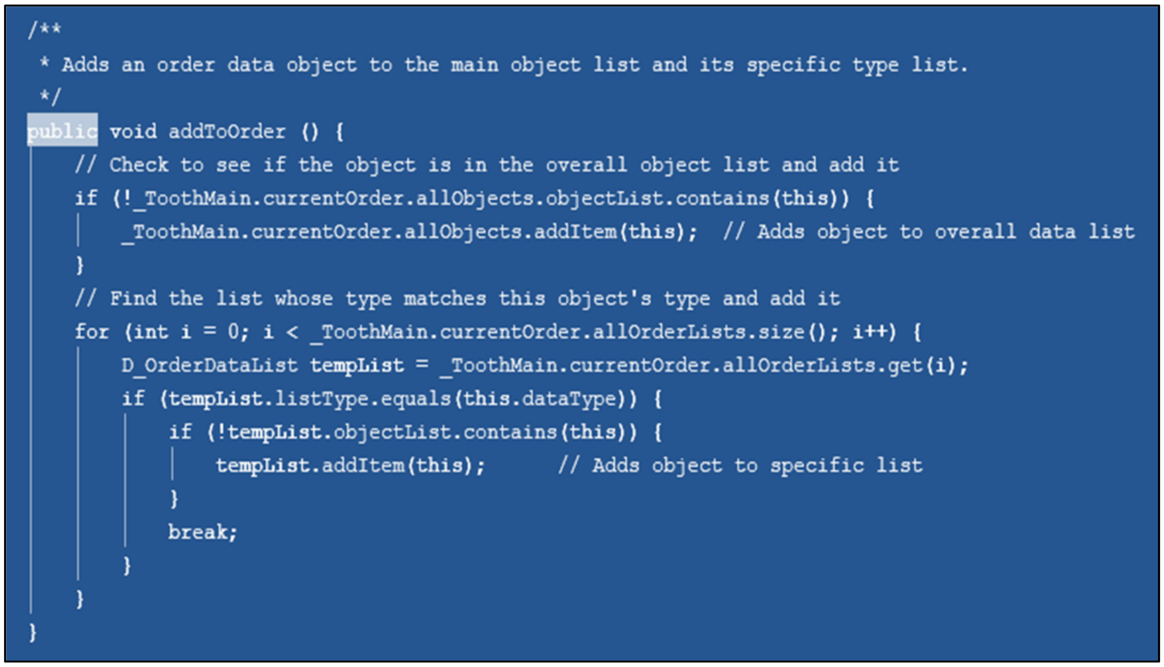
a. Object Storage and Referencing

As discussed in Paragraph III.C, each individual piece of information in the operations order was implemented either as its own unique object (if the data relationship graph indicated that it could be used as a parameter for another information object) or as an attribute to one of the unique information object classes. These objects all contain a “type” attribute that denotes the appropriate list where they should be categorized within the greater OrderData object. All object types inherit the properties and methods (functions) of the root OrderDataObject class. One of these methods is called addToOrder() and its purpose is to ensure that each new data object created is filed in the appropriate list within the OrderData object. The pseudocode for this function is given in Figure 29. For comparison, a screen capture of the actual Java code for the same function is shown in Figure 30.



```
Algorithm Add to Order
Input: None
Output: None
if object is not in order data list of all objects:
    add object to the list of all objects

for each list in the order data list of lists:
    if list type == this data object's type:
        if item is not in specific list:
            add item to specific type list
            break
```

Figure 29. Pseudocode for Add to Order Function.



```java
/**
 * Adds an order data object to the main object list and its specific type list.
 */
public void addToOrder () {
    // Check to see if the object is in the overall object list and add it
    if (!_ToothMain.currentOrder.allObjects.objectList.contains(this)) {
        _ToothMain.currentOrder.allObjects.addItem(this);  // Adds object to overall data list
    }
    // Find the list whose type matches this object's type and add it
    for (int i = 0; i < _ToothMain.currentOrder.allOrderLists.size(); i++) {
        D_OrderDataList tempList = _ToothMain.currentOrder.allOrderLists.get(i);
        if (tempList.listType.equals(this.dataType)) {
            if (!tempList.objectList.contains(this)) {
                tempList.addItem(this);      // Adds object to specific list
            }
            break;
        }
    }
}
```

Figure 30. Screen Capture of Java Code for Add to Order Function.

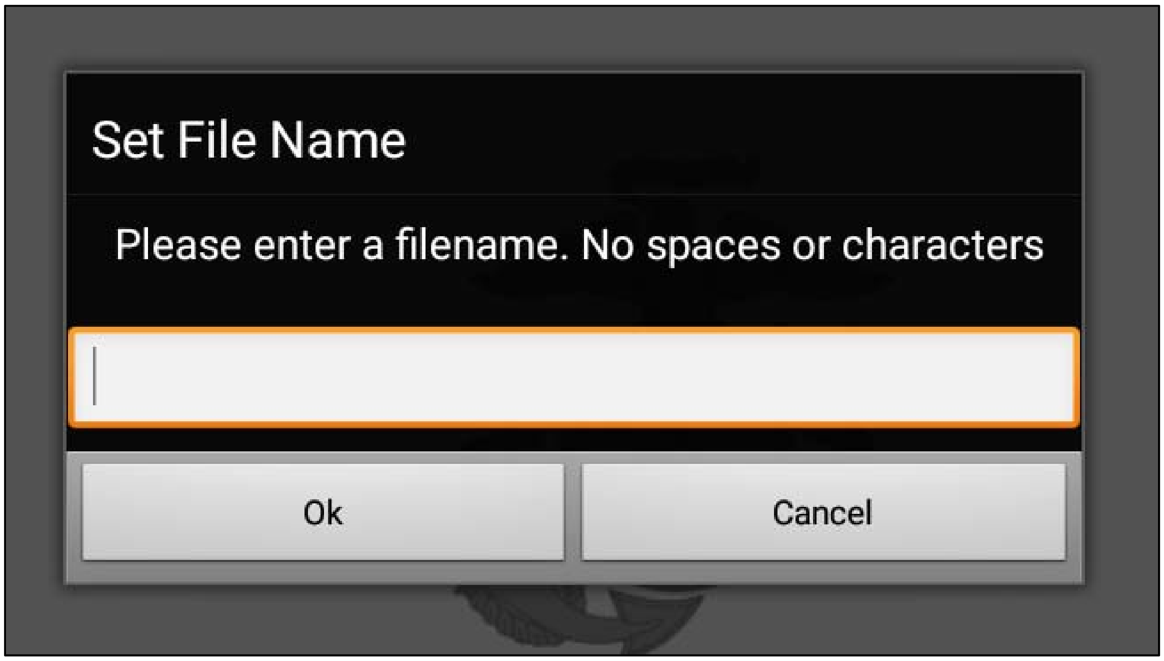
A similar function exists for the removal of objects from the operations order. Unlike when objects are added, however, the removal functions must ensure that the object to be deleted is not referenced by any other object. If, for example, the unit object to be removed is linked as the parent unit in the attributes of another unit object, then it must first be removed from the second unit object before being removed from the order as a whole. This requires each object to have a custom deletion method that iterates through all potentially linked objects and severs those links before calling the universal removal method of the root data class. Once the object to be removed is no longer linked to any other object and is not referenced in the organization lists maintained by the OrderData container, it is no longer accessible by the application and is subsequently removed by the Java garbage collector.

b. Data Saving and Reloading

Saving whole or selected portions of the data structure to the device's storage is relatively trivial due to the fact that all of the objects in the data structure are serialized as discussed in Paragraph 1.a. This allowed us to save portions of the data structure as templates that can be reused in different orders, which is a helpful feature for leaders whose unit organizations do not change very often. We used the same construct in each of these cases:

1. The user requests to save the order or a template via a button in the user interface.
2. The user is prompted to provide a meaningful string for use as a file name. This file name is then checked for validity and appended with an appropriate file extension.
3. TOOTH queries the operating system's file manager to see if the specified file name already exists in the TOOTH data directory. If so, it prompts the user to select a different file name.
4. A copy of the selected data is serialized and the byte stream is written to a file with the selected file name.
5. The file is closed and the application waits for the next user input.

Figure 31 shows TOOTH prompting a user to select a filename.


Set File Name

Please enter a filename. No spaces or characters

Figure 31. Screen Capture of Filename Prompt.

Loading saved data follows a similar construct in reverse, with the notable difference that the data being loaded replaces the data in the current order. This requires examining the current data structure and breaking links and removing objects that are destined to be replaced. Assume, for example, that a user created an order with a single unit and started assigning tasks to that unit before remembering that he could save time by importing a previously-used unit structure. When he clicks to import a saved unit template, the application will ask him to acknowledge that his action will remove his existing unit. Once he clicks to accept that action, the application iterates through the list of current units and calls their self-removal methods before prompting him to select a saved template.

c. Order Focus Shifting

Both the unit hierarchy and the task elements of the data structure are extensible, meaning that their tree-like structure can be expanded to fit various levels of operations orders. This concept is the key behind the goal of making the application "focusable"—that is, able to shift the focus of an operations order from one unit to another. As discussed in Chapter II, the current methods of creating an operations order from a higher headquarters or adjacent unit order consists of manually identifying and copying information from the existing order to the appropriate places in the new order.

Rather than shuffle information around, TOOTH simply shifts the focus from one element of an extensible structure to another. When a user selects the option to create a new order from an existing order, the application presents a list of saved orders for her to choose from. The order that she chooses is loaded into the active order placeholder, and the user is then presented a list of units within that order (with the exception of the currently-assigned base unit). The user selects a new base unit, and the application simply changes the base unit designation for the order from the previous unit to the new unit. All of the information related to the previous base unit is still there, but now a different set of links between information objects takes priority. For example, if the user selected a company-sized unit based on a battalion's order, she should expect that the battalion commander's intent is still resident in the operations order. Now, however, instead of that intent being categorized as a sub-paragraph of the Execution section of the order, it is now categorized as "higher's intent" in the Situation paragraph. Nothing about the commander's intent or the relationship of the two units changed, and viewing the extensible structure from a different angle merely caused information to shift in prominence.

C. USER INTERFACE

No matter how well the data structure worked, TOOTH would be a failure without an effective interface between the information objects and the user manipulating them. The creation of this interface turned out to be a more ambitious project than originally planned, as the implementation easily consumed 75% or more of the time spent programming the application. This section explores the process of creating that interface.

1. Fragment Interactions

The primary building blocks of an Android user interface are Java classes called "fragments" that define how the application responds to user interaction with the interface. Each fragment has a corresponding XML file called a "layout" that defines the visual elements of the fragment and specifies the spatial relationships between those visual elements (Morris 2011). The fragment initiates a series of "listeners" that are tripped if the operating system registers that specific events have occurred. If a user clicks an on-screen button, that click registers with Android's onClickListener() function, and associates the click with a unique button identifier. The operating system then executes any code that is listed in the fragment's onClick() method. This code can include anything from displaying a dialogue box to performing arithmetic to manipulating the data structure, or even launching another fragment.

Every application invokes an Android class known as a "fragment manager" that handles the creation, tracking, and deletion of all fragments in the application. When a fragment is created, the function that called for its creation also specifies a target "frame" for it to fill. These frames can be thought of us window panes—they can be as large as the whole screen, or as small as the designer wishes them to be. It is the fragment manager's job to measure the designated frame, specify the sizing of different objects in the layout file based on the space available, and finally declare the fragment to be visible once its elements have all been drawn to the screen (Google 2017).

Fragments can be nested inside of each other, which also requires the creation of nested fragment managers. This technique is used in TOOTH to allow user selections in one portion of the interface to present different content views in another portion of the interface. Figure 32 demonstrates the nesting of fragments within the main application screen.

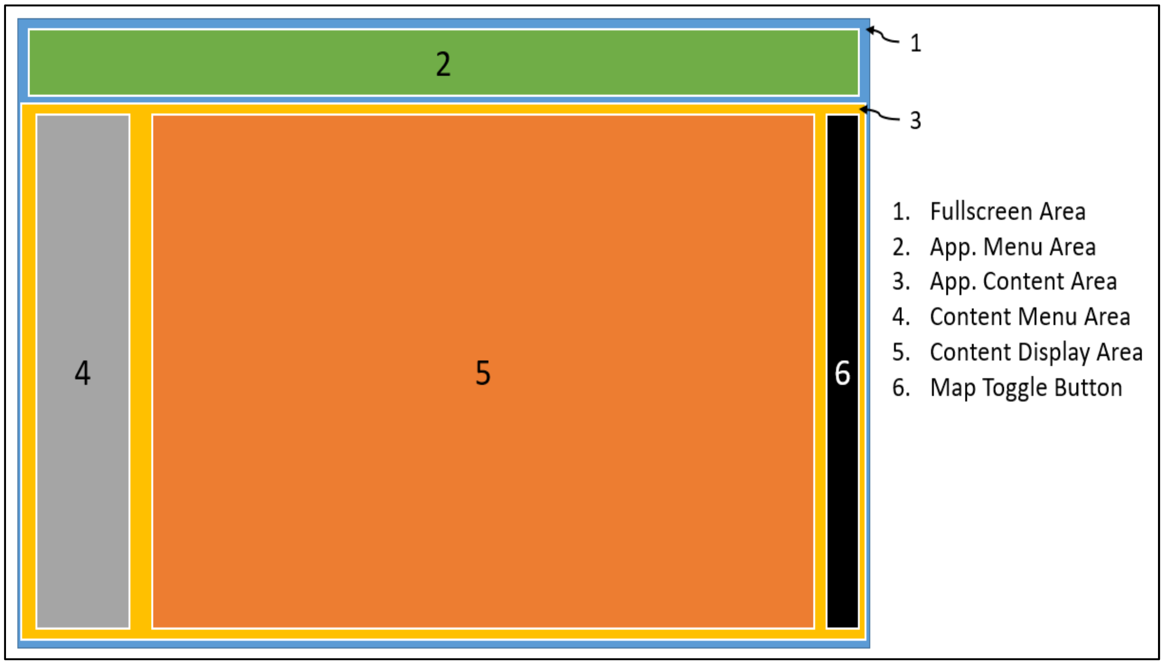


Figure 32. Fragment Area Nesting in TOOTH.

The fullscreen area is divided into two parts—the menu ribbon across the top and the content area across the majority of the bottom. The content area implements its own "child" fragment manager that controls the opening and closing of fragments in the content menu, content display, and map toggle button areas. The actual implementation of this same nesting is shown in Figure 33.

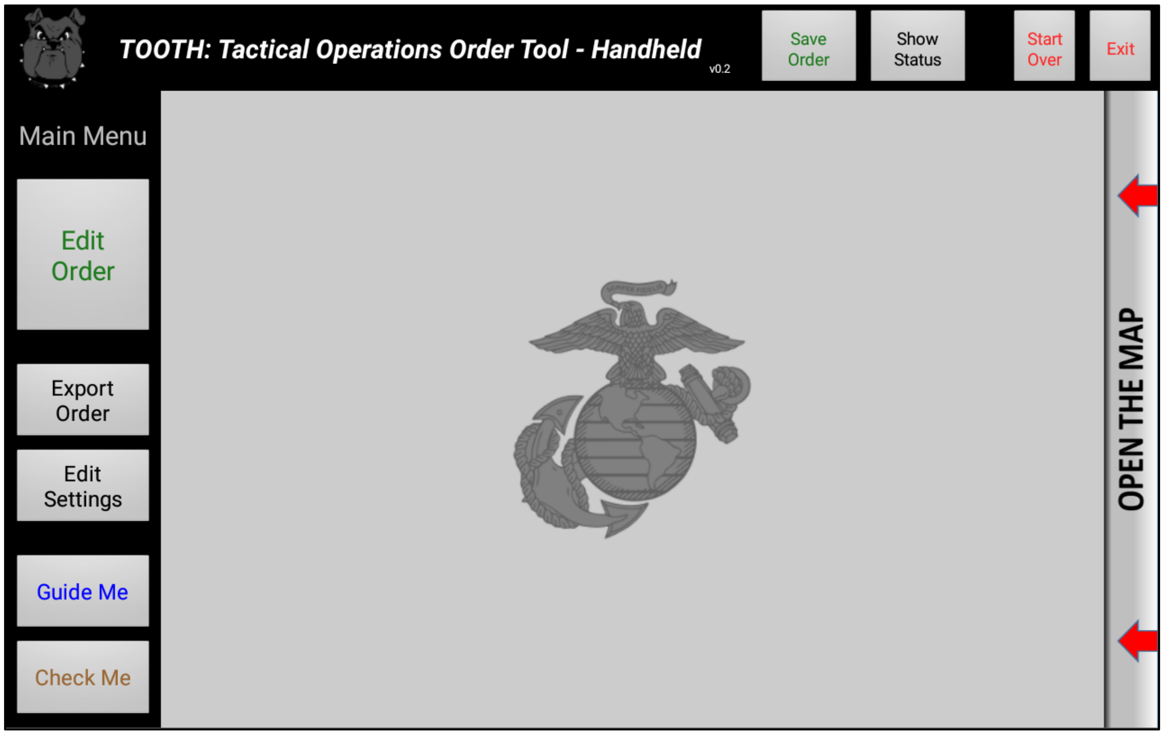


Figure 33. TOOTH User Interface.

Buttons on the application menu bar across the top of the screen generally change the entire screen view, whereas buttons in the left side menu change the content displayed in the central portion of the screen and also adjust the left side menu to provide context for the user's location within the menu tree.

Nearly every menu selection in TOOTH causes a fragment to be created, hidden, or made visible. In order to avoid repeating similar blocks of code for every button definition, we instead built a set of helper functions tied to the child fragment managers for the major elements of the user interface. These helper functions keep track of the currently-visible fragments and provide the ability to load new fragments with a single line of code.

2. Map Implementation

The original goal of building TOOTH on top of the KILSWITCH map technology proved to be infeasible, as discussed in Paragraph A.2. This was problematic because the ability to conduct graphically-based planning and immediately see the effects of information changes on a map is a core capability of TOOTH. This left two options: implement different mapping software or create a series of static fragments and images to show how the technology might look when implemented. The latter option would have been easier to build, but would ultimately have distracted viewers from being able to visualize the technology's ability to present multiple ways to view the same information (to include viewing changes in either "side" of the app as they are made).

a. OpenStreetMap

We examined two software solutions in order to ensure that TOOTH would contain a representative mapping capability. The first was GoogleMaps, for which Google provides an API to allow developers to incorporate maps into their applications. GoogleMaps is simple to implement, but does not provide an option for developers to integrate their own custom-generated offline mapping (saving maps to the storage of the device for reference without an internet connection). Instead, TOOTH implements OpenStreetMap, an open-source clone of the GoogleMaps software. The API for OpenStreetMap provides a number of the features included in our design requirements, including the ability to use offline mapping, the ability to add touch controls (such as "pinch zoom") to the map interface, and the ability to build multiple custom overlays of clickable icons for display on selected maps.

Each map instance of the OpenStreetMap software is its own fragment. TOOTH uses one instance for the tactical map and creates and destroys overlays that appear on that instance, rather than generating a new instance every time the map portion of the user interface is loaded. This helps to ensure that changes made by the user regarding map positioning and zoom level are not undone each time the user navigates back and forth between the map and the rest of TOOTH's user interface.

The remainder of the desired map functions not provided by OpenStreetMap were implemented by building a custom, transparent user interface fragment that sits on top of the OpenStreetMap fragment. The Android API allows users to click through a transparent background and interface with the layer beneath, which allowed us to place "floating" buttons above an otherwise clickable map. The full map interface is displayed in Figure 34.

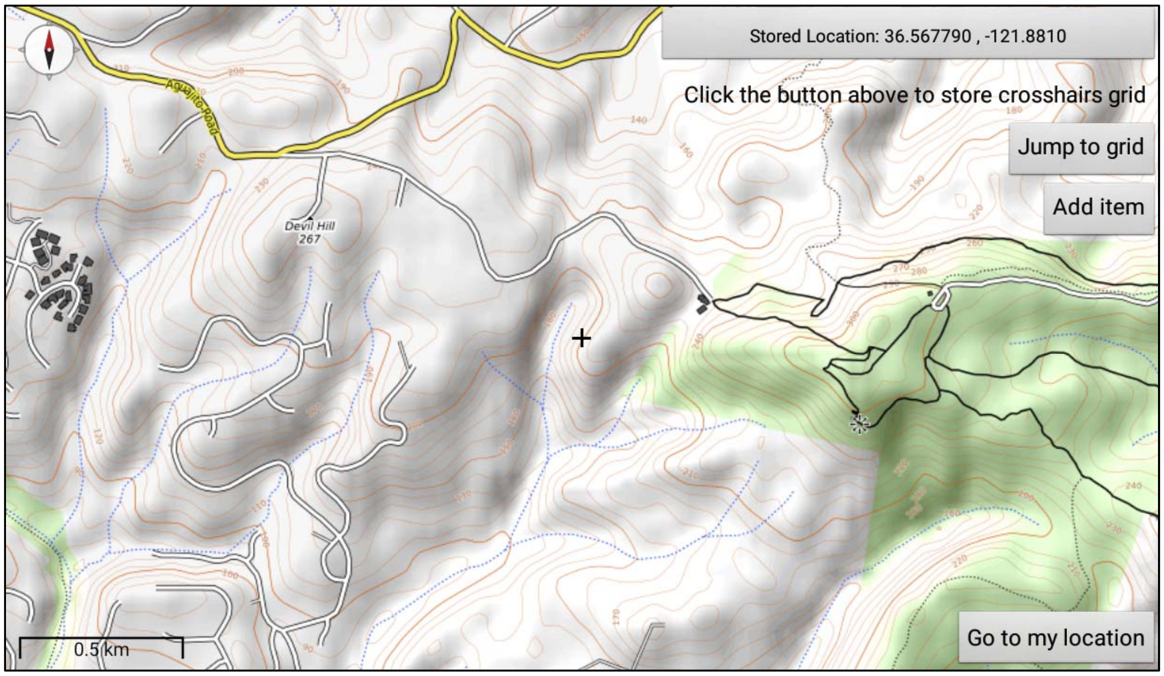

Stored Location: 36.567790 , -121.8810
Click the button above to store crosshairs grid
Jump to grid
Add item
Go to my location
Devil Hill 267
Aguajito Road
0.5 km

Figure 34. Full Map Interface.

The "Go to my location" button queries the device's GPS and centers the map on that location. The "stored location" button allows users to capture the coordinates that are underneath the crosshair on the center of the screen for use in editing items. The "jump to grid" button allows the user to input coordinates on which to center the map. Finally, the "Add item" button allows the user to add a data element (unit, TCM or task) at the stored coordinates.

b. Overlay Generation

Order data items are plotted on the map using three types of overlays: unit locations, TCM locations, and task locations. Whenever a user adds, deletes, or edits an information object contained in one of the overlays, that version of the overlay containing the item is flagged for update the next time that the user accesses the mapping interface. If the user adds or edits an item from the mapping interface itself, the change is made immediately. This prevents the application from devoting un-necessary processing cycles to map updates if the user is changing a whole series of data points in one of the non-map user interfaces. An example of a non-topographical mapping layer with a friendly unit overlay plotted is shown in Figure 35.

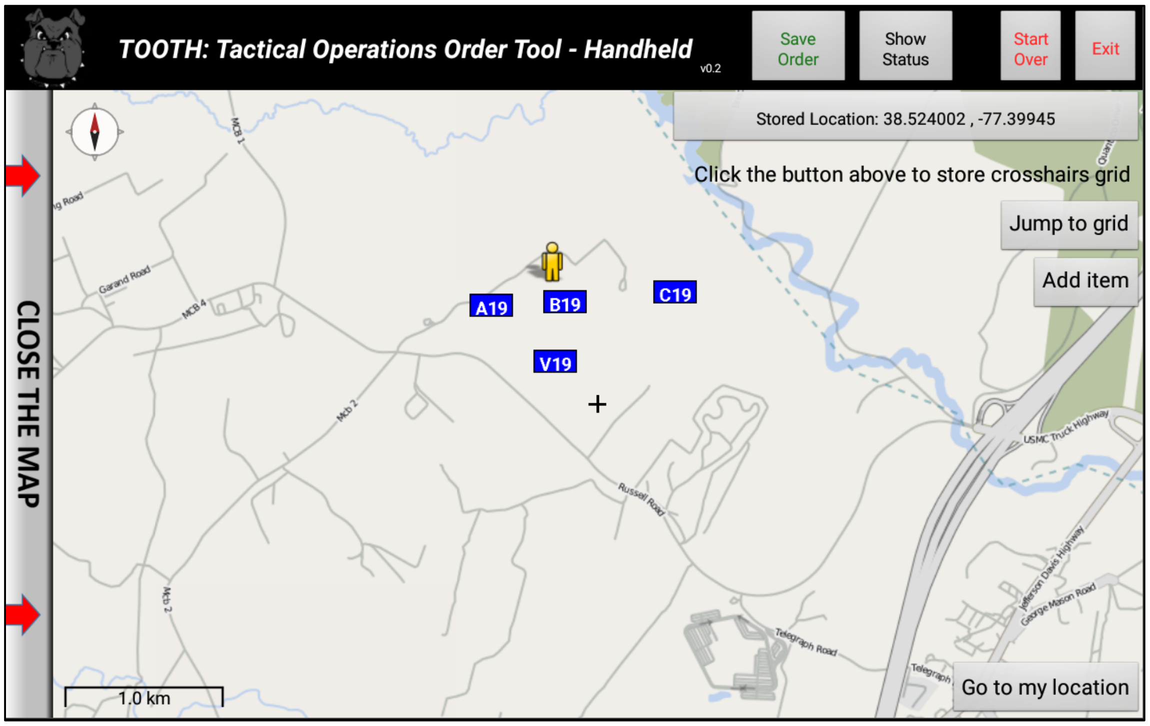

TOOTH: Tactical Operations Order Tool - Handheld
v0.2
Save Order
Show Status
Start Over
Exit
Stored Location: 38.524002 , -77.39945
Click the button above to store crosshairs grid
Jump to grid
Add item
Go to my location
CLOSE THE MAP
A19
B19
C19
V19
1.0 km

Figure 35. Map Interface with Unit Location Overlay.

3. Data Entry and Display Implementation

Chapter III discussed the need for the data manipulation portions of the user interface to be as simple and intuitive as possible. This section discusses the methods that were used to make that goal a reality.

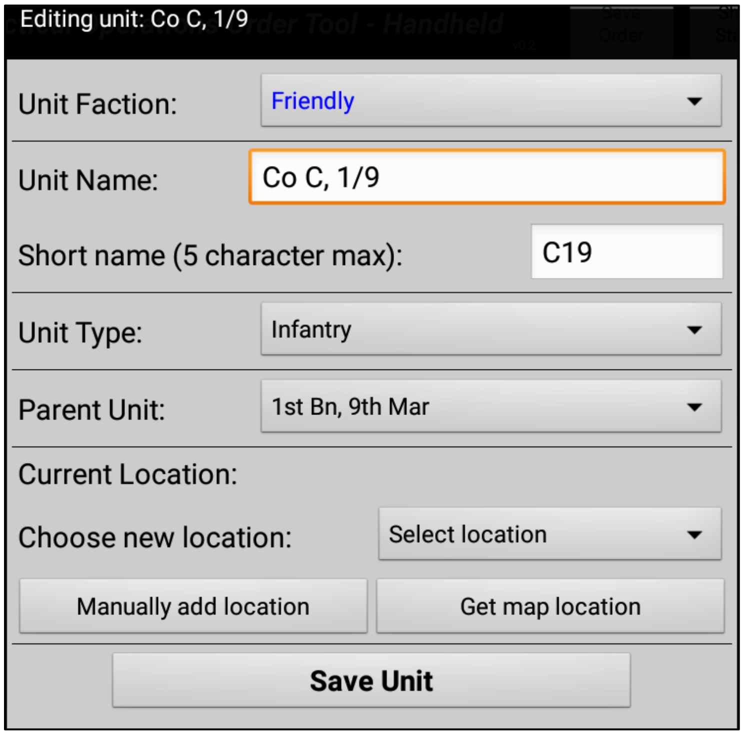
a. Data Input Forms

The central idea of this research is that the information in an operations order can be represented as a series of interconnected objects. Thus, the majority of the data manipulation in the interface is done by presenting the user with forms and lists of eligible objects. These lists are created using an Android method called an ArrayAdapter that turns a list array of objects into a clickable user interface (Google 2017). The list of items presented to the user can be customized to display whichever object attribute text is most appropriate. When the user selects an item, its position in the list corresponds with its index in the original array of objects, allowing the selected object itself to be linked as an object attribute in a one-to-one or one-to-many relationship. An example function that uses an ArrayAdapter is shown in Figure 36.

```
Algorithm selectParentUnit
Input: A list index
Output: Sets a unit object's parent unit attribute
previous parent unit = null
previous parent unit = unit being edited parent unit

new dialog with title "select parent" and adapter set to parent adapter:
    selected parent unit = null
    unit to find (string) = selected adapter item string
    for each item in list of all units:
        temp unit = item
        if item name == unit to find:
            selected parent unit = temp unit
            break

    test (Boolean) = checkTreeForConflicts() to prevent unit tree loops

    if test == false:
        if previous parent = false:
            parent unit = selected parent unit
        else:
            replaceParent() function severs child link in previous parent and sets new
        set button text to new parent
    else display an error and prompt for a different selection
    close dialog
```

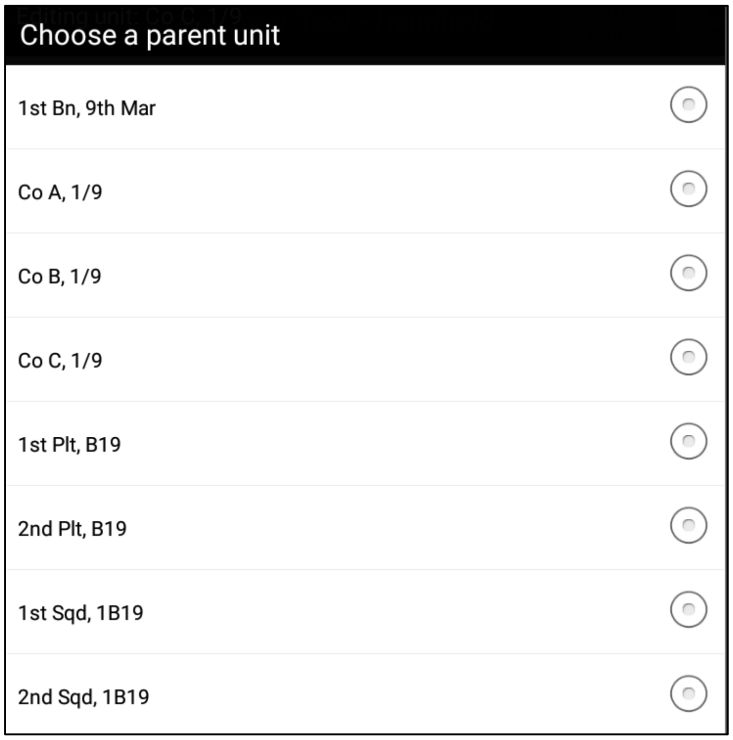
Figure 36. Pseudocode for Select Parent Unit Array Adapter Usage.

This function is called by the form for editing a unit's information when the user presses the "parent unit" button. The form is shown in Figure 37 and the ArrayAdapter interface for this function is shown in Figure 38.


Editing unit: Co C, 1/9

Unit Faction:	Friendly
Unit Name:	Co C, 1/9
Short name (5 character max):	C19
Unit Type:	Infantry
Parent Unit:	1st Bn, 9th Mar
Current Location:	
Choose new location:	Select location

Figure 37. Example Form for Unit Object Editing.


Choose a parent unit

- 1st Bn, 9th Mar
- Co A, 1/9
- Co B, 1/9
- Co C, 1/9
- 1st Plt, B19
- 2nd Plt, B19
- 1st Sqd, 1B19
- 2nd Sqd, 1B19

Figure 38. ArrayAdapter Interface for Parent Unit Selection.

b. Unit Hierarchy View

Although the majority of the data entry and manipulation can be handled through the use of forms and lists, there are several information structures that are best represented in a manner that graphically depicts the relationships between information objects. The unit hierarchy is one of those structures, as discussed in Paragraph III.D.3. At its heart it is a simple tree structure, with each unit object containing attributes for linking to a parent unit object and a list of child unit objects. The soundness of the graph is maintained by checking for loops prior to allowing a unit to be assigned as the child of another unit. This is done by recursively scanning a unit's child tree for references to the proposed parent before formally establishing the relationship. The real implementation dilemma was how to best display the information in a way that is intuitive to the user but also flexible to immediate change. The author's vision was for a self-drawing organizational chart that presented options to the user when units were "clicked" in the user interface. A search for open-source organizational chart-style graph generation code did not yield any results, and so the focus of effort shifted to creating a custom solution.

The author experimented with creating a custom "canvas" that would update after any changes to organizational structure, but quickly realized that the bitmapping code required to make individual unit graphics "clickable" required more than a trivial effort. Instead, the author realized that he could use the same OpenStreetMap software used for the mapping interface to create custom overlays of units on a "blank" map. Like the graphics in the map interface, these icons could be programmed with menu options to be presented to the user depending on his or her actions. Figure 39 shows the unit hierarchy interface with a single battalion-level unit created.

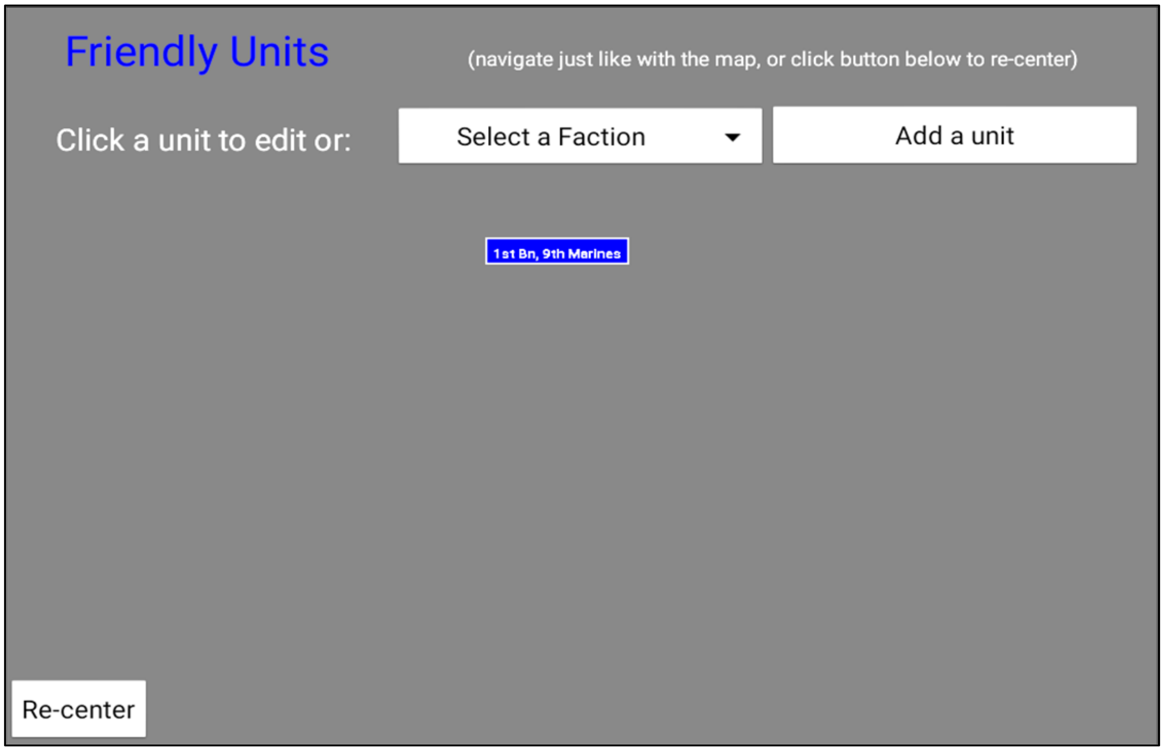


Figure 39. Unit Hierarchy Interface Fragment.

Clicking on the unit provides basic information about the unit, while clicking and holding presents several options to the user, as demonstrated in Figure 40.

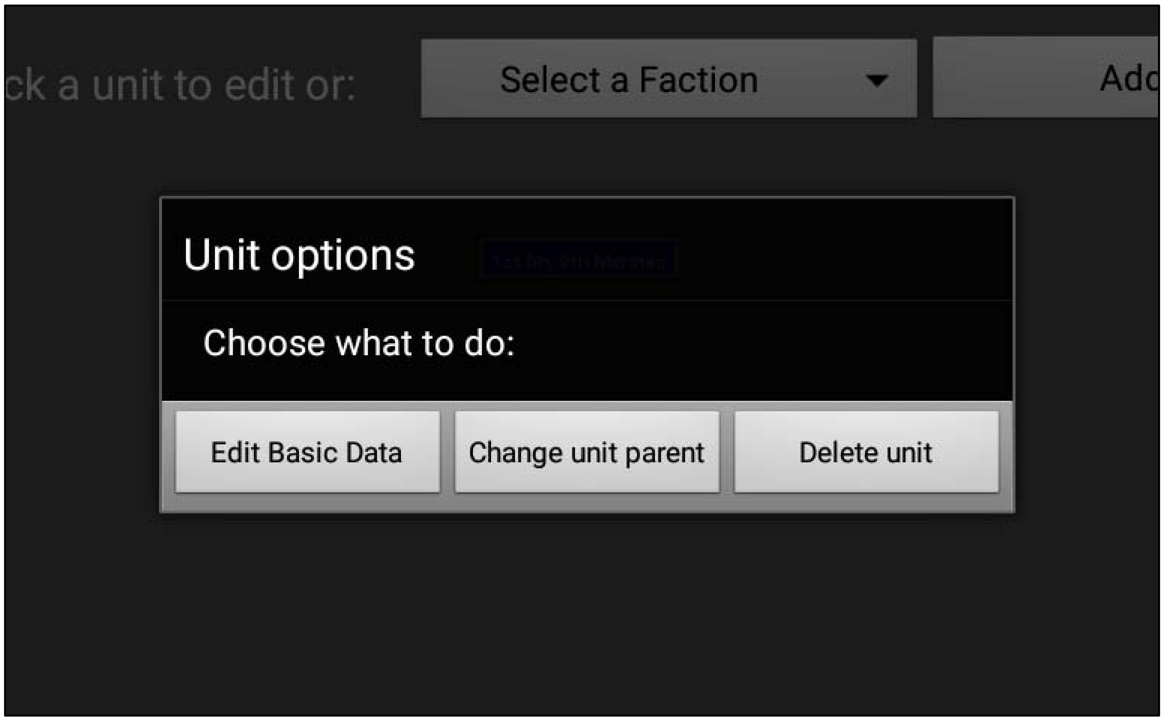
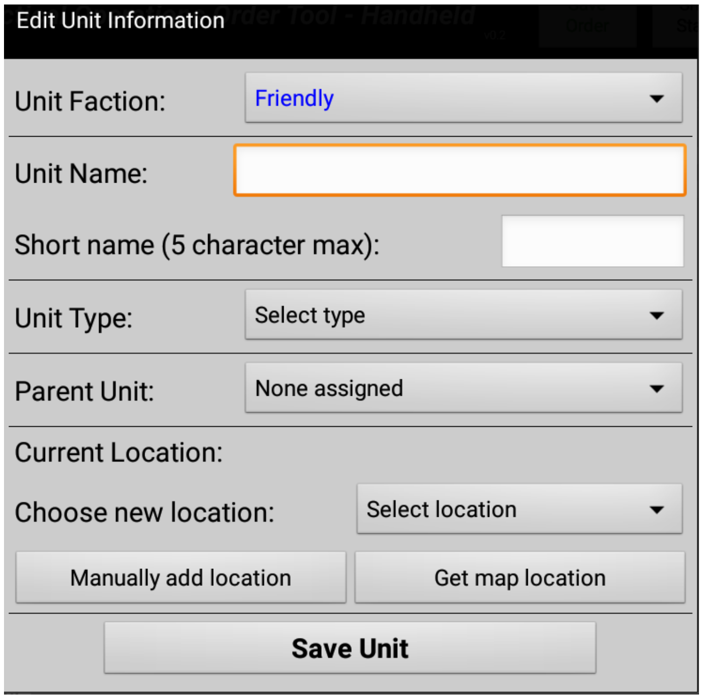


Figure 40. Unit Hierarchy Interface – Unit Editing Option Menu.

When the user clicks the button to add an additional unit, she is presented with another data entry form as depicted in Figure 41. This is the same form used to edit an existing unit’s information. In that case, the unit’s currently-saved information is pre-populated into the form before it is presented to the user.


Edit Unit Information

Unit Faction:	Friendly
Unit Name:	
Short name (5 character max):	
Unit Type:	Select type
Parent Unit:	None assigned
Current Location:	
Choose new location:	Select location

Figure 41. Unit Information Form Fragment.

As with previously-discussed forms, the drop-down menus in this form use array adapters to link to previously-created information objects.

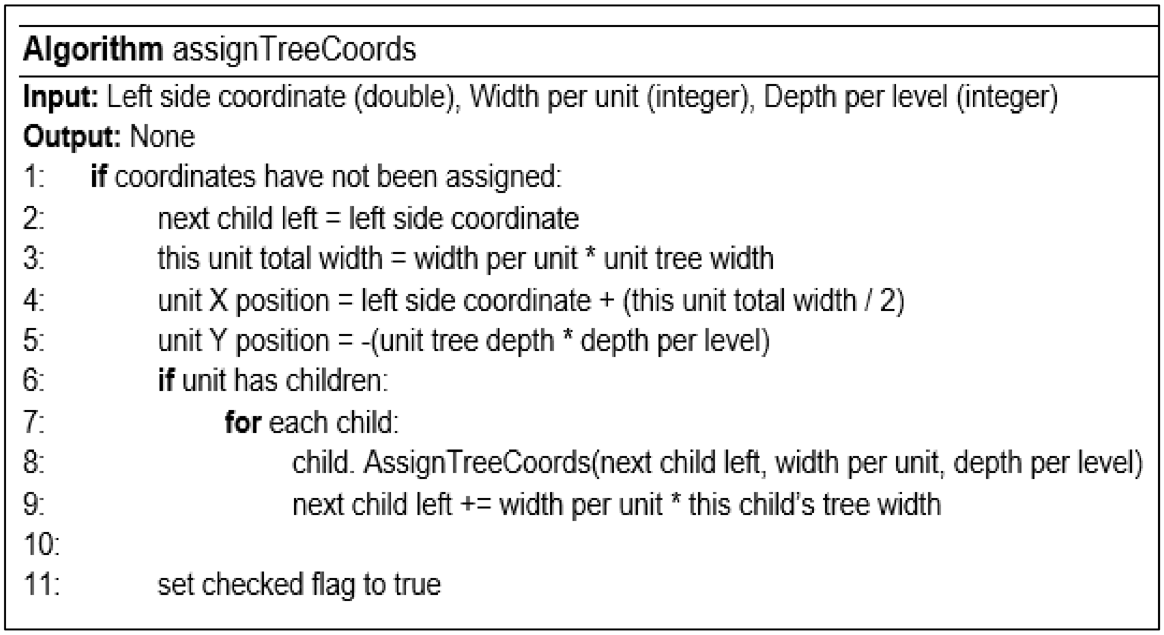
Once the user clicks to save a unit, TOOTH must decide how to display the changes in the user interface. This required the creation of a two-tiered algorithm to assign display coordinates to each unit (its relative position on the “blank” map), followed by algorithms to create the unit icons, place them into map overlays, and draw connecting lines to show the structure of the tree. The first step in the coordinate-generation is to recursively determine the structure of its child tree using a depth-first search that starts from parent-less root node units. This determines how much horizontal space each section of the unit hierarchy tree will consume. The pseudocode for this step is shown in Figure 42.

```
Algorithm getTreeWidths
Input: The graph depth of the parent unit (integer)
Output: Max width (in units) of the child tree (integer)
if unit has already been checked:
    present an error message
    break
else:
    set checked flag to true

if object graph depth is >= 0:
    present an error message
    break
else:
    set object graph depth = parent depth + 1

if current tree width >= 0:
    present an error message
    break
else:
    temporary tree width = 0
    if unit has children:
        for each child:
            temporary tree width += child tree widths (recursive)
    else:
        temporary tree width = 1
    unit tree width = temporary tree width

return unit tree width
```

Figure 42. Pseudocode for Unit Hierarchy Tree Width Calculations.

Once each unit knows how large its tree of children is, the next step is to assign all units a set of mock-Cartesian coordinates. These coordinates are determined relative to one another in an X/Y plane using fixed widths and heights for node depth and width, but are ultimately converted to actual latitude/longitude coordinates for plotting on the “blank” world map. (The actual center of the unit hierarchy interface is centered several hundred miles south of Ghana at the intersection of the equator and the prime meridian, the 0/0 location in the mock-

Cartesian system). The pseudocode for assigning these coordinates is given in Figure 43.



```
Algorithm assignTreeCoords
Input: Left side coordinate (double), Width per unit (integer), Depth per level (integer)
Output: None
if coordinates have not been assigned:
    next child left = left side coordinate
    this unit total width = width per unit * unit tree width
    unit X position = left side coordinate + (this unit total width / 2)
    unit Y position = -(unit tree depth * depth per level)
    if unit has children:
        for each child:
            child. AssignTreeCoords(next child left, width per unit, depth per level)
            next child left += width per unit * this child's tree width

    set checked flag to true
```

Figure 43. Pseudocode for Unit Hierarchy Coordinate Assignment.

Once each unit has been re-assigned coordinates, they are re-plotted on a map overlay on top of horizontal and vertical lines that use each unit's coordinates and relationships to show the links between units. The graph resulting after adding several levels of subordinate units is shown in Figure 44.

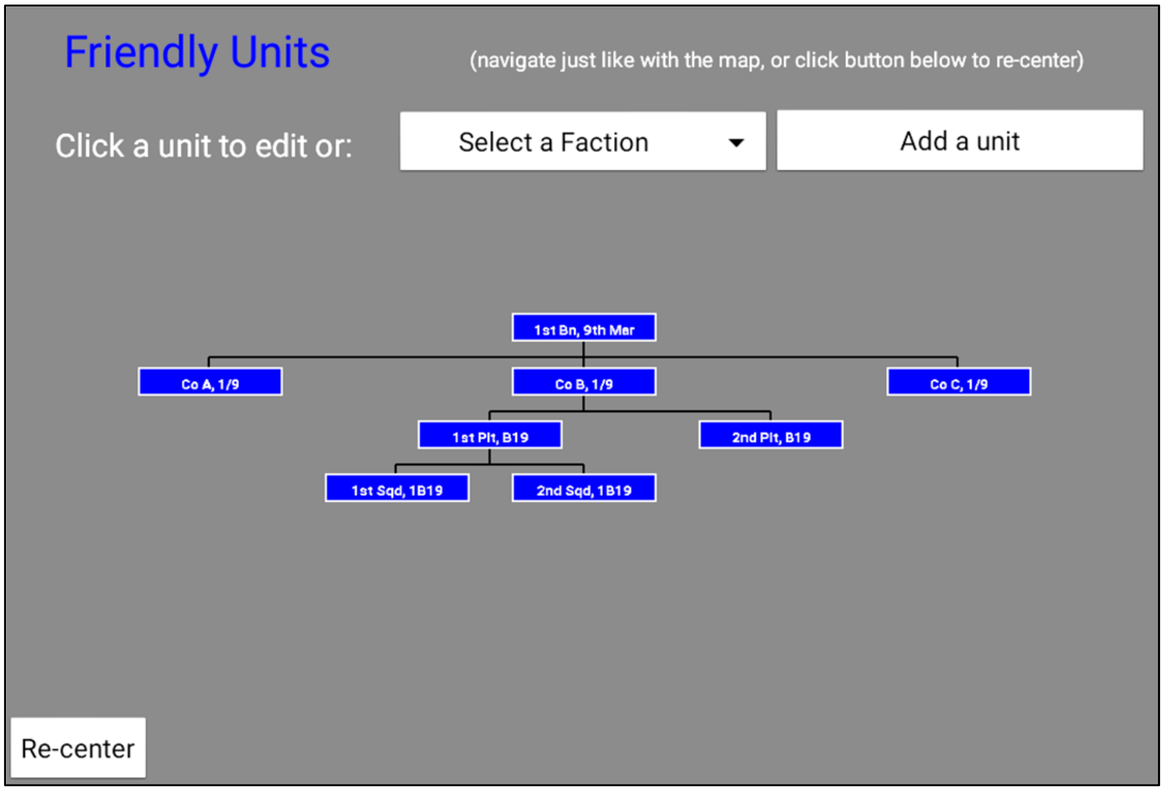
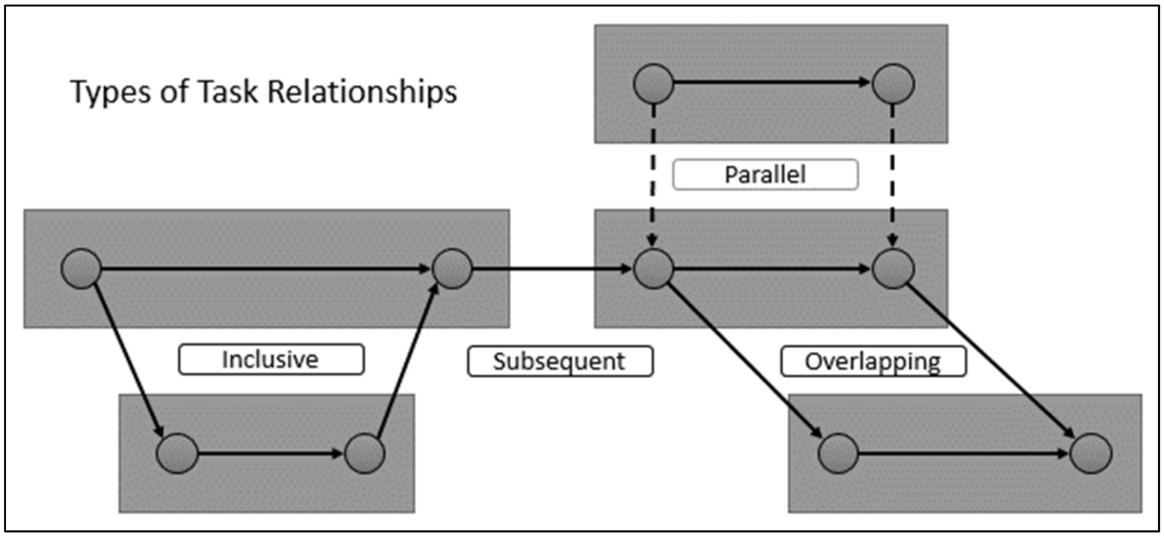


Figure 44. Unit Hierarchy Interface With Multiple Unit Levels.

c. Task Dependency View

Displaying the relationships between tactical tasks presented a similar interface challenge and prompted a similar solution. As discussed in Paragraph III.C.3, the proper ordering of tactical tasks is central to a well-constructed operational plan, but displaying those relationships in an intuitive and dynamic user interface presented a difficult problem. The structure of task dependencies can be modeled as a complicated directed acyclic graph (DAG), for which there are well-established algorithms (Cormen, Leiserson, and Rivest 2001). These algorithms ensure consistency within the graph, measure distance between nodes, and perform other useful functions. As with the unit hierarchy, we could not find any useful and available Android Java code sources for creating dynamic DAGs. We developed our own implementation, once again using OpenStreetMap to provide a dynamically-updatable and clickable user interface.

Several factors make a DAG of tactical task relationships more complicated than one for other processes such as civilian project timelines. Not all dependencies between tactical tasks are simple cause-and-effect relationships where one task must be completed before others can be started, an easier problem to graph and one suited for a traditional Gantt chart. Some tasks must overlap with others, such as when machine gun fire suppresses an objective so that infantrymen can maneuver onto the enemy position. Some tasks mirror the execution timelines of another task, such as when a unit is tasked to follow-in-support of another unit. In the latter case, the timeline for the task to follow-in-support is dependent on the primary task that it is supporting. In order to allow for overlaps, parallel timelines, and inclusionary tasks (where one task must start after another task but finish before that task is complete), we split each task into start and end nodes and used the relationships between those nodes to define the relationships between tasks. This concept is shown in Figure 45.



Solid arrows between nodes show which node must come “first” in the ordering of the DAG. Dashed arrows indicate nodes that are tied to the DAG ordering of other nodes (parallelism).

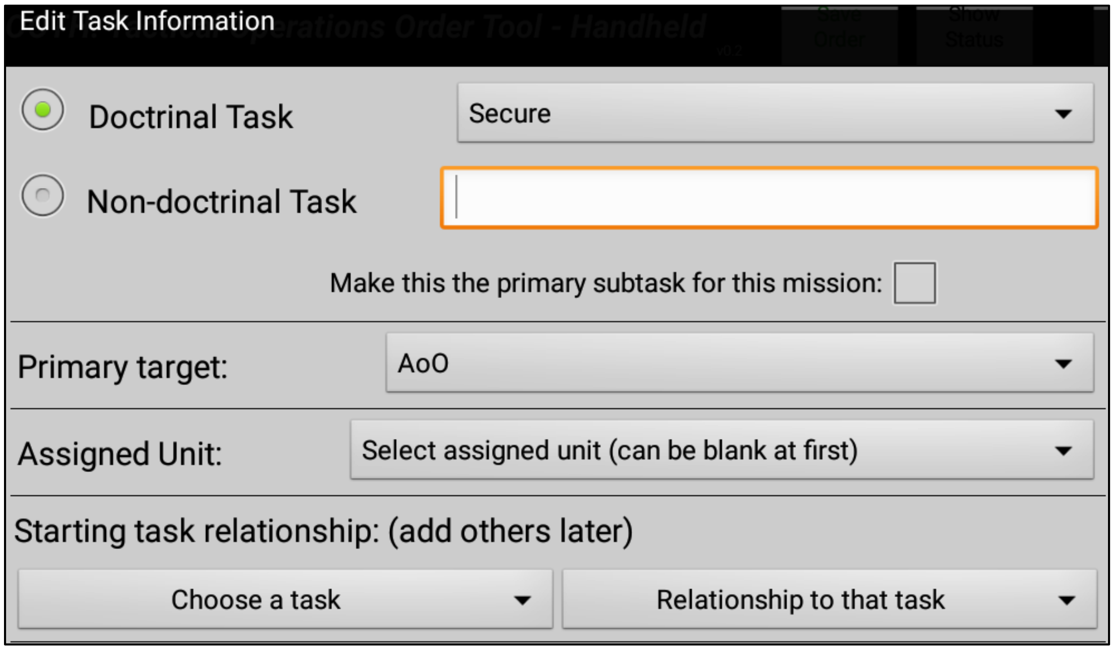
Figure 45. Examples of Task Relationships.

As with the unit hierarchy interface, TOOTH checks for graph loops at the time that a user attempts to add a relationship. This ensures that the graph can be ordered and placed on an arbitrary timeline. The actual process of assigning relative positions to nodes is adapted from the DAG-SHORTEST-PATH procedure (Cormen, Leiserson, and Rivest 2001, 593) and is shown in Figure 46.

```
Algorithm graphTasks
Input: A list of inter-connected task nodes
Output: An ordered graph of task objects
(find the root nodes)
for node in list of nodes:
    if node is not in parallel with another node:
        if node has no precursor nodes:
            add node to list of root nodes
    else:
        add to list of parallel nodes to assign later

(find longest chain / earliest root)
for node in root nodes:
    find length of successor node chain (recursive) – Bellman-Ford algorithm

(start numbering recursively at root node with longest chain)
remove node from remaining list of root nodes
set its depth number to 1
for each subsequent linked node:
    if depth number is < current node depth + 1:
        set its new depth number to current node depth + 1
        recursively re-assign its successors
    else:
        ignore successor node

(assign remaining nodes backwards, in order of highest minimum successor value)
while there are unassigned nodes:
    for node in unassigned node list:
        find lowest depth number of successive nodes
        if lowest successive node > lowest successor of current next node to assign:
            designate this node as next node to assign
    set node to assign depth = lowest successor node depth - 1

(assign values to simultaneous nodes)
for node in parallel node list:
    set depth = parallel node depth
```

Figure 46. Pseudocode for Task Dependency Graph Ordering. Adapted from Cormen, Leiserson, and Rivest (2001).

Once the start- and end-nodes are assigned an ordering, it is a simple problem to assign them X coordinates on a Cartesian plane. In order to compress the resulting interface into a screen with minimal scrolling, TOOTH utilizes an algorithm which “walks” the graph from left to right and assigns task nodes the highest available vertical “row” based on whether or not their graphic would overlap with a previously-drawn graphic. The menu interface for adding or editing a task is shown in Figure 47 and the full task dependency interface is shown in Figure 48.


Edit Task Information

(●) Doctrinal Task — Secure

( ) Non-doctrinal Task —

Make this the primary subtask for this mission: ☐

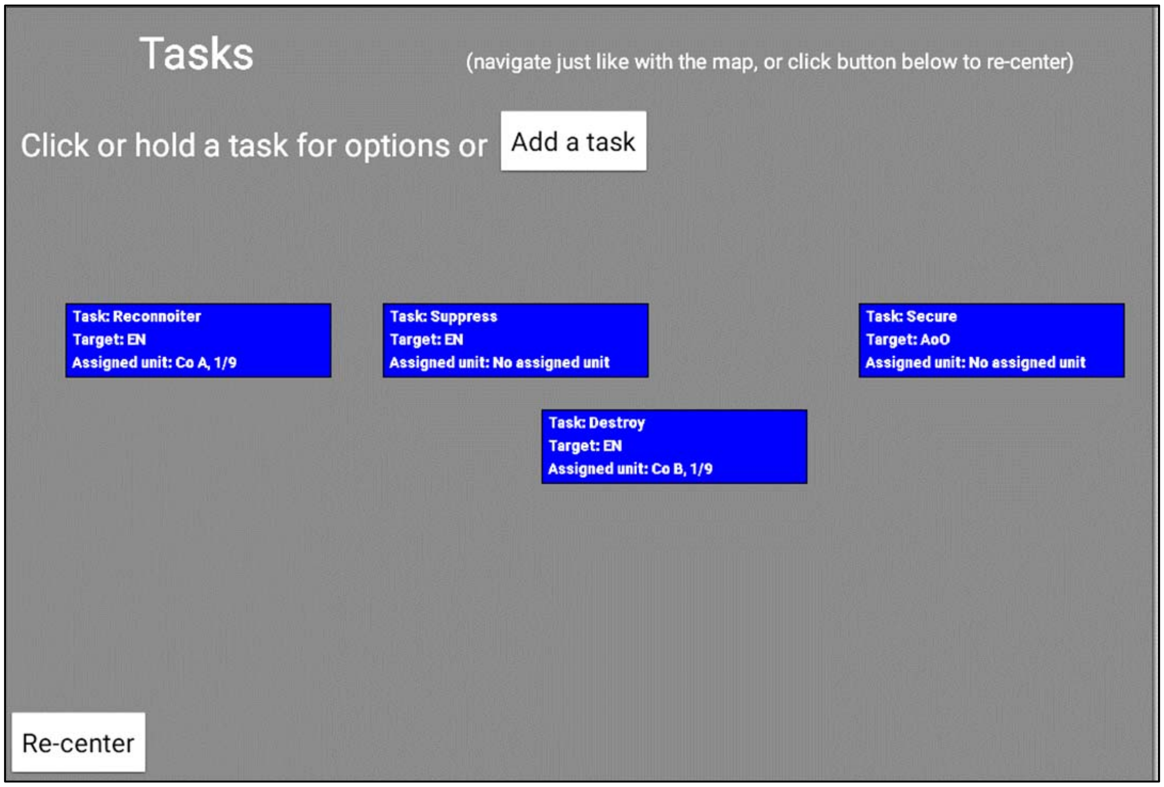
Primary target: AoO

Assigned Unit: Select assigned unit (can be blank at first)

Starting task relationship: (add others later)

Choose a task — Relationship to that task

Figure 47. Task Editing Form Interface.


Tasks
(navigate just like with the map, or click button below to re-center)
Click or hold a task for options or
Add a task
Re-center

Task: Reconnoiter
Target: EN
Assigned unit: Co A, 1/9

Task: Suppress
Target: EN
Assigned unit: No assigned unit

Task: Secure
Target: AoO
Assigned unit: No assigned unit

Task: Destroy
Target: EN
Assigned unit: Co B, 1/9

Figure 48. Task Dependency Interface.

d. HTML Plain-Text View

One of the last functions developed for the technology demonstrator is a Hyper-Text Markup Language (HTML) generator that demonstrates the potential to create any number of specifically-formatted documents based on the TOOTH data structure. This function arranges the human-readable attributes of the order information objects into a form that approximates the traditional Five Paragraph Order Format. Using HTML allows for the resulting document to be displayed within a custom HTML-viewing fragment in TOOTH or on any other device with a web browser. Furthermore, this exported text document could be imported into more-powerful word processing software for conversion to other formats or inclusion into slide shows.

D. DEVELOPMENT STATISTICS

In the course of implementing the features listed above, the author generated three major revisions of the user interface and twelve major milestones in the application structure. Each of the features implemented was tested incrementally, resulting in upwards of three hundred total software builds. At the time of publication, the author had personally generated 61 Java files containing more than 8,430 lines of code and 74 XML files containing an additional 6,490 lines. This code relies on Android and OpenStreetMap libraries consisting of another ~336,000 lines of code. An archive file containing the project source code is included as an attachment to this thesis.

E. DEMONSTRATION

The technology has been successfully demonstrated at its current maturation level to several entities within the Marine Corps. The author and his advisors travelled to the Washington, D.C., area in February of 2017, where they met with NPS research sponsors as part of a broader synchronization of research efforts. Part of these meetings included the opportunity to use the technology demonstrator to brief potential application stakeholders on the central ideas of this thesis. TOOTH was discussed in several meetings at The Basic School, the Infantry Officer Course, and with Marine Corps innovation leaders at MCB Quantico and the Pentagon. The overall response to the idea was very positive, especially among those with first-hand experience creating operations orders, and it is a testament to the strength of the concept that everyone who saw the demonstration immediately started thinking about what the idea meant for their particular communities, saying “it would be great if it could do ...”

The technology was also demonstrated to various Marine Corps entities at the 2017 NPS Naval Research Working Group (NRWG), with several expressing interest in the concept. One Marine Corps research sponsor agreed to pursue further topics related to the TOOTH concept. This connection helps set the stage for follow-on research described in Chapter V.

F. SUMMARY

This chapter detailed the creation of the TOOTH data structure and user interface. It covered aspects of the implementation in depth, especially where custom algorithms or solutions were required in order to achieve the goals of the technology demonstrator. Chapter V presents conclusions about the work completed to-date and suggest ways that the capability of TOOTH can be enhanced through future research work.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS, RECOMMENDATIONS AND FUTURE WORK

This chapter summarizes the results of this research in the form of observations on the performance of the technology demonstrator. It also presents ideas for improving the central themes of this research through future work.

A. CONCLUSIONS

Although the efficiency and usability of TOOTH was not formally tested in the course of this research, we believe that the implementation of the central ideas of this thesis has shown them to be valid and worthy of further study and eventual adoption by the Marine Corps and other services. The following observations are arranged according to the characteristics of an ideal application presented at the end of Chapter II:

1. *Mobile*. The software utilizes features of the Android operating system to effectively harness the capabilities of modern mobile devices, and the technology demonstrator application runs effectively on a hand-held Samsung Galaxy Tab device.

2. *Valid*. We believe that the data structure provides a methodology for accurately representing the attributes of the different types of information in an operations order and provides example methods for defining the connections between those information objects.

3. *Usable*. We believe that the user interface, while simple, is intuitive and is capable of presenting the information in the operations order to the user in ways that allow the user to effectively manipulate the data.

4. *Focusable*. TOOTH shows the immense time-saving potential of the object-oriented approach to organizing the information resident in tactical operations orders. The process of creating a new operations order shell from an existing higher-headquarters order, with all of the previous information reformatted into its new locations (a process that takes between five minutes and an hour to complete using manual methods), is accomplished accurately and completely within TOOTH in less than ten seconds.

5. *Exportable*. The application's successful ability to save and re-generate data objects implies a wide range of potential data export techniques. Furthermore, the initial work related to generating plain-text HTML documents from the order's information indicates the potential for the order information to be manipulated by a natural-language algorithm for rapid conversion to human-readable formats that are familiar to all Marines (Kiser 2016).

6. *Assistive*. The simple options menus implemented in TOOTH show the potential for a more robust algorithm to guide the user through the orders generation process while simultaneously checking that order for potential mistakes.

In summary, we believe that the technology demonstrator produced for this thesis effective communicates the potential presented by its ideas and provides a structural framework upon which to build.

B. RECOMMENDATIONS

The following recommendations are based on observations made during the completion of this thesis.

(1) Continued Development and Testing

The TOOTH application is matured to the point that it is capable of demonstrating the central ideas of this thesis, but in its current state it is far from a complete or useable product. The author intends to continue working toward completion of an actual prototype during his follow-on assignment, and would be happy to help any future NPS students who are interested in research contained in the following future work section.

We recommend that any development of the technology prior to wide distribution in the Marine Corps or other services include a complete re-working of the code by professional Android Java programmers. Ideally, this development effort would include subject-matter experts whose understanding of the tactical operations order generation process would help to focus and clarify the efforts of the technical professionals.

(2) Marine Corps Adoption

We believe that this technology has the potential to revolutionize the way that tactical operations orders are created and disseminated, but we also know that the barriers to service-wide adoption are high. The best course of action is to engage the stakeholder organizations that have a say in the doctrine and training of tactical operations orders. Targeting new generations of leaders who are more familiar with the use of mobile technology will help to demonstrate the time saving potential to traditionalists.

(3) NPS Code Repository

The NPS computer science department does not have a central code repository for archiving and sharing code produced by students in the course of their thesis work. In addition to the potential loss of work product as students graduate, the lack of a central repository potentially complicates the efforts of students who are working on joint theses or continuing the research of previous students. Furthermore, it introduces the possibility of potential leaks of NPS code via open commons licenses when students resort to using commercial version control products. We believe that NPS would benefit from the creation and maintenance of a central code repository that is advertised to all students as a forum for sharing algorithms.

C. FUTURE WORK

A review of Appendix A reveals that many of the envisioned functions of TOOTH are incomplete. Several of these are strong avenues for future research in applying computing principles in artificial intelligence, natural language programming, and simulator development to the central idea of this thesis.

(1) Expert System Generation

The basic menu options and user interactions programmed into TOOTH show the potential for quickly guiding users to accomplish their intended actions, but are barely a starting point for the possible ways that an expert system could help users with the creation and dissemination of operations orders.

- “Guide Me” Function

One potential research project would be the application of artificial intelligence principles of expert systems to the full process of guiding a user through the generation of an operations order. This would include the use of order type templates to shape the user’s selection of appropriate TCMs and tactical tasks, and would include the guided creation of sound schemes of maneuver by walking the user through doctrinally-correct relationships between tactical tasks.

- “Check Me” Function

Another potential expert system would check the order for errors in the user’s tactical thought process that extend beyond issues that are easily fixed by the design of the data structure. Examples of this type of functionality include simple problems such as assigning a unit tactical tasks whose completion is impossible due to constraints of time and space, or more complex issues such as finding errors in an operation’s geometry of fires by using data on unit locations and their associated weapon systems.

(2) Natural Language Output

The simple HTML-generation tool included in the technology demonstrator is sufficient to demonstrate the ability to convert object parameters into exportable plain text; it is not sufficient in its current form for producing human-readable plans from the interconnect information objects in a TOOTH operations order file. Natural language techniques could be used to automatically export the operations order information in formats familiar to all basically-trained recipients.

(3) Simulator (BML) Output

The potential also exists to export TOOTH data in a format that is machine-readable for generating simulated versions of live operations orders.

This is a potential research area for a modeling and simulation student to use TOOTH as a tool for converting tactical thought into Battle Management Language or other XML formats, especially if those products can be used with the Marine Air-Ground Task Force (MAGTF) Tactical Warfare Simulation (MTWS) (Marine Air Ground Task Force Training Center 2017).

(4) Application Effectiveness Testing

The success of the function that creates a new operations order from an existing order suggests that TOOTH could improve user efficiency by as much as 90% or more in certain steps of the orders generation process. Assumptions like that need to be verified with actual human-subject testing, preferably in an environment such as a formal school that uses operations orders in its curriculum and provides the opportunity for a large sample size and a control population.

(5) Doctrinal Completeness Review

The data types and templates included in the technology demonstrator are sufficient to demonstrate the manipulation of inter-connected data, but are not an exhaustive or doctrinally-complete set of default data types. Future work on TOOTH should continue to expand the default options presented to users in order to ensure that the platform is sound and produces relevant operations orders.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. GOAL HIERARCHY LIST

Tables 1 through 6 represent the full goal hierarchy list used in creating TOOTH. The implementation column denotes whether or not the goal was fully implemented in TOOTH prior to publication. The Java code files for major functions are listed in the source file column. Implemented goals with no associated file name are implemented within the file name listed under their nearest parent goal.

Table 1. Goal One Hierarchy List.

Goal	Implemented?	Source File
G1. Initiate application	Yes	_ToothMain.java
1.1. Load user interface	Yes	_ToothMain.java
1.1.1. Cover loading process with splash screen	Yes	A_SplashActivity.java
1.1.2. Load interface components	Yes	_ToothMain.java
1.1.2.1. Set up fragment managers	Yes	
1.1.2.2. Load main content	Yes	
1.1.2.3. Load map engine	Yes	
1.1.3. Load data templates	Yes	D_OrderData.java
1.1.3.1. Check for default object lists	Yes	
1.1.3.1.1. Create if missing	Yes	
1.1.4. Load menu items	Yes	M_MainMenu.java
1.2. Set a user	Yes	M_UserMenu.java
1.2.1. Populate user ID field	Yes	I_CreateUserObject.java
1.2.2. Create new user	Yes	I_CreateUserObject.java
1.2.2.1. Generate user ID	Yes	
1.2.2.2. Save user in user file	Yes	
1.2.2.3. Save preferences in user file	Partial	
1.2.3. Load a saved user	Yes	L_UserList.java
1.2.3.1. Open user file	Yes	
1.2.3.2. Allow user to select ID	Yes	
1.2.3.3. Validate user with 4-digit code	No	
1.2.4. Delete existing user	Partial	M_SettingsMenu.java
1.2.4.1. Validate user privilege to delete account	No	
1.3. Load preferences and libraries	Yes	M_SettingsMenu.java
1.3.1. Import preference list	Partial	M_SettingsMenu.java
1.3.2. Import libraries (and build libraries if they don't exist)	Yes	D_OrderData.java
1.3.2.1. Import order type list	Partial	
1.3.2.1.1. Allow user creation of new types	No	
1.3.2.2. Import task type list	Yes	
1.3.2.2.1. Allow user creation of new types	No	
1.3.2.3. Import tactical control measure type list	Yes	
1.3.2.3.1. Allow user creation of new types	No	
1.3.2.4. Import unit type list	Yes	
1.3.2.4.1. Allow user creation of new types	No	
1.3.2.5. Import faction list	Yes	
1.3.2.5.1. Allow user creation of new types	Partial	
1.3.3. Import saved unit hierarchies	Yes	M_SettingsMenu.java
1.3.3.1. Designate order unit in import hierarchy	Yes	

Table 2. Goal Two Hierarchy List.

Goal	Implemented?	Source File
G2. Generate new operations order	Yes	D_OrderData.java
2.1. Designate order type (used in error checking)	Partial	D_OrderType.java
2.1.1. Load default order types	Partial	
2.2. Initiate data structure	Yes	D_OrderData.java
2.2.1. Initiate base order information class	Yes	
2.3. Prompt for user guiding (see G6)	Partial	_ToothMain.java
2.4. Input battlespace framework information	Partial	M_EditPara1Menu.java
2.4.1. Input areas	Partial	
2.4.1.1. Create area of operation (unique)	No	
2.4.1.2. Create area of influence (unique / optional)	No	
2.4.1.2.1. Set equal to AoO?	No	
2.4.1.3. Create area of interest (unique / optional)	No	
2.4.1.3.1. Set equal to AoO?	No	
2.4.1.4. Add to operations overlay	Partial	
2.4.1.4.1. Add to map overlay display	Partial	
2.4.2. Input friendly unit hierarchy	Yes	C_UnitTree.java
2.4.2.1. Input friendly unit	Yes	I_AddUnitObjectDialog.java
2.4.2.1.1. Input unit name	Yes	
2.4.2.1.1.1. Designate display name (5 char limit)	Yes	
2.4.2.1.2. Input unit location	Yes	
2.4.2.1.2.1. Get location from map input	Yes	
2.4.2.1.2.2. Get location from manual input	Yes	
2.4.2.1.3. Input unit type	Yes	
2.4.2.1.4. Input unit relationships	Yes	
2.4.2.1.4.1. Designate parent unit	Yes	
2.4.2.1.4.2. Designate child unit	Yes	
2.4.2.1.4.3. Designate support relationship	No	
2.4.2.1.5. Input weapons system types	No	
2.4.2.1.5.1. Load weapons systems from library	No	
2.4.2.1.5.2. Store weapons information as template	No	
2.4.2.1.6. Input unit communications information	No	
2.4.2.1.6.1. Input number / type of radios	No	
2.4.2.1.6.1.1. Call information from library	No	
2.4.2.1.6.2. Input communications nets / callsigns	No	
2.4.2.1.6.2.1. Generate radio freq/netID list	No	
2.4.2.1.6.2.2. Generate radio freq/netID datatypes	No	
2.4.2.1.7. Add to operations overlay	Yes	
2.4.2.2. Prompt to save unit hierarchy as template	Yes	
2.4.3. Input enemy units	Yes	C_UnitTree.java
2.4.3.1. Input unit name	Yes	I_AddUnitObjectDialog.java
2.4.3.1.1. Designate display name (5 char limit)	Yes	
2.4.3.2. Input unit location	Yes	
2.4.3.2.1. Get location from map input	Yes	
2.4.3.3. Input unit type	Yes	
2.4.3.4. Input unit relationships	Yes	
2.4.3.4.1. Designate parent unit	Yes	
2.4.3.4.2. Designate child unit	Yes	
2.4.3.4.3. Designate support relationship	Yes	
2.4.3.5. Add to operations overlay	Yes	

Table 3. Goal Two (Continued) and Goal Three Hierarchy Lists.

Goal	Implemented?	Source File
2.4.4. Input Tactical Control Measures (TCMs)	Partial	
2.4.4.1. Input objectives	Partial	
2.4.4.1.1. Add to operations overlay	Partial	
2.4.4.1.1.1. Add to map overlay display	Partial	
2.4.4.2. Input checkpoints	Partial	
2.4.4.2.1. Add to operations overlay	Partial	
2.4.4.2.1.1. Add to map overlay display	Partial	
2.4.4.3. Input phase lines	Partial	
2.4.4.3.1. Add to operations overlay	Partial	
2.4.4.3.1.1. Add to map overlay display	Partial	
2.5. Input unit mission statement (Paragraph 2 – Mission)	Yes	M_EditPara2.java
2.5.1. Designate a unit in hierarchy as owning unit for the order	Yes	M_EditPara2.java
2.5.2. Create task tree with this task as its base	Yes	C_TaskTree.java
2.6. Input scheme of maneuver	Partial	M_EditPara3Menu.java
2.6.1. Create tasks	Yes	C_TaskTree.java
2.6.1.1. Designate associated TCM(s)	Partial	
2.6.1.2. Designate task dependency	Yes	
2.6.1.3. Designate task concurrency	Yes	
2.6.2. Create non-task timeline events	Yes	
2.6.2.1. Indicate event dependency	Yes	
2.6.2.2. Indicate event concurrency	Yes	
2.6.3. Re-order tasks and events	Partial	
2.6.4. Designate phases/stages/parts	Partial	
2.6.5. Assign tasks to subordinate units	Yes	
2.7. Input fire support plan	Partial	
2.7.1. Create fires tasks	Yes	
2.7.1.1. Tie fires task to supported maneuver tasks	Yes	
2.8. Input administration and logistics information (paragraph 4)	Partial	M_EditPara4Menu.java
2.8.1. Input non-auto generated information	Partial	
2.9. Input command and signal information (paragraph 5)	Partial	M_EditPara5Menu.java
2.9.1. Input non-auto generated information	Partial	
G3. Open existing operations order	Yes	L_SavedOrderList.java
3.1. Confirm file password	No	
3.2. Decrypt file	No	
3.3. Read file	Yes	
3.3.1. Populate data structures	Yes	

Table 4. Goals Four Through Seven Hierarchy Lists.

Goal	Implemented?	Source File
G4. Generate new order from existing order	Yes	L_ModifiableOrderList.java
4.1. Designate base unit from existing order	Yes	D_OrderData.java
4.1.1. Peek at existing unit structure	Yes	
4.1.2. Designate base unit	Yes	
4.1.3. Update unit mission	Yes	
4.1.3.1. Create task tree with this task as its base (See G2)	Yes	
4.2. Open existing order (see G3)	Yes	
4.2.1. Use designated base unit for situation and task import	Yes	
4.3. Update orientation / situation information	Yes	
4.3.1. Import tactical control measures	Yes	
4.3.1.1. Confirm area of operations	Yes	
4.3.2. Import higher unit information	Yes	
4.3.2.1. Update higher mission and intent	Yes	
4.3.3. Import adjacent unit information	Yes	
4.3.4. Import supporting unit information	Yes	
4.3.4.1. Update fires availability matrix	No	
4.3.5. Input additional battlespace framework information (see G1)	Partial	
4.4. Input scheme of maneuver information	Partial	
4.4.1. User confirmed clear of current data	Partial	
4.4.1.1. User confirmed clear of tasks for subordinate units	Partial	
4.4.1.2. User confirmed clear of fires information	Partial	
4.4.2. Task and event input and sequencing (see G2)	Yes	
4.4.3. Fires input and sequencing (see G2)	Yes	
G5. Edit existing order	Yes	L_SavedOrderList.java
5.1. Open existing order (see G3)	Yes	
5.2. Allow user editing of existing data structures (see G2)	Yes	
5.3. Prompt for user guiding (see G6)	Partial	
G6. Guide user through order creation	Partial	
6.1. Update this based on process flow determination (task tree)	No	
6.2. Prompt for auto-generation of additional products (see G7)	No	
6.3. Prompt for order validation (see G8)	Partial	
G7. Auto-generate additional order sections and products	No	
7.1. Populate administration and logistics portion	No	
7.1.1. Generate TCM list	No	
7.1.2. Generate task-specific PCC / PCI list	No	
7.2. Populate command and signal portions	No	
7.2.1. Generate signal list from ordered SOM list	No	
7.2.2. Generate frequency / call sign list from unit hierarchy	No	
7.2.3. Generate succession of command list (ordered list)	No	
7.2.3.1. Allow user addition of additional persons	No	
7.2.3.2. Allow user deletion of persons	No	
7.2.3.3. Allow user re-ordering or list	No	
7.2.3.4. Allow for re-generation of default	No	
7.3. Populate additional products	No	

Table 5. Goal Eight Hierarchy List.

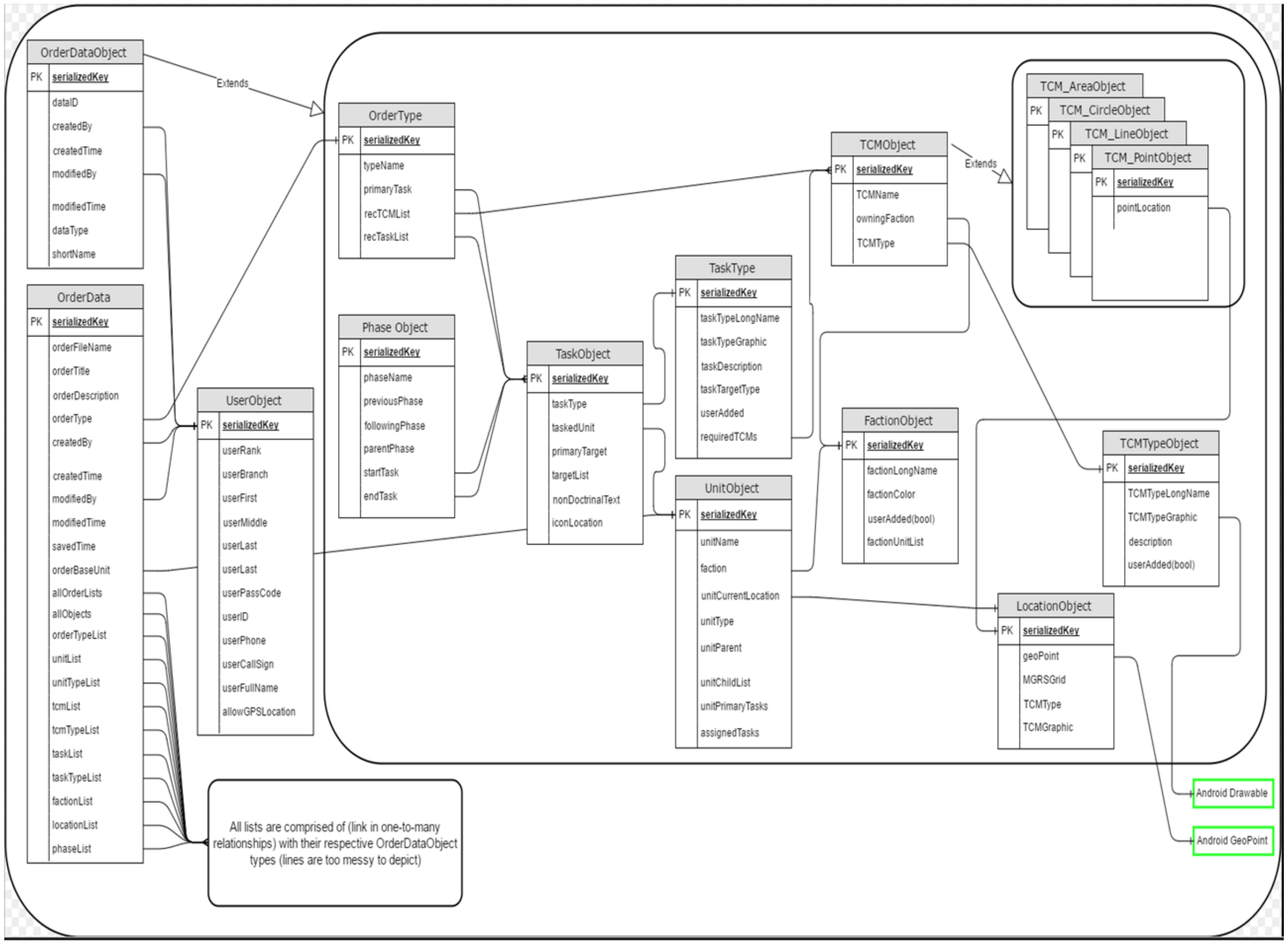
Goal	Implemented?	Source File
G8. Validate order	No	
8.1. Confirm situation / orientation validity (para 1)	No	
8.1.1. Confirm no duplicate TCM names	No	
8.1.1.1. Check adjacent unit checkpoint names	No	
8.1.1.2. Check duplicate objectives on same grid	No	
8.1.1.2.1. Combine object names in data type	No	
8.1.2. Confirm TCMs are inside of area of operations	No	
8.1.3. Check validity of friendly unit tree	No	
8.1.3.1. Confirm no unit double-attached	No	
8.1.3.2. Confirm all unit parent/support relationships	No	
8.2. Confirm mission validity (para 2)	No	
8.2.1. Confirm mission tied to unit	No	
8.2.2. Confirm mission tied to appropriate objective	No	
8.3. Confirm execution validity (para 3)	No	
8.3.1. Confirm task validity	No	
8.3.1.1. Confirm all tasks tied to units	No	
8.3.1.2. Confirm all supporting tasks tied to primary task	No	
8.3.1.3. Confirm all tasks have signal associated	No	
8.3.1.4. Confirm no unit has more than one task per phase	No	
8.3.1.5. Confirm task location suitability	No	
8.3.2. Confirm SOM validity	No	
8.3.2.1. Confirm SDZ geometry	No	
8.3.3. Confirm fires validity	No	
8.3.3.1. Confirm supporting assets firecap prior to use	No	
8.3.3.2. Confirm supporting assets in range	No	
8.3.3.3. Confirm round counts support timeline	No	
8.4. Confirm administration and logistics validity (para 4)	No	
8.5. Confirm command and signal validity (para 5)	No	
8.5.1. Confirm signal plan validity	No	
8.5.1.1. Confirm each task / event has a primary signal	No	
8.5.1.2. Confirm each task / event has a secondary signal	No	
8.5.1.3. Confirm units on same net are within frequency range	No	
8.5.1.4. Confirm units have enough radios to maintain comms	No	
8.5.2. Confirm command information validity	No	
8.5.2.1. Confirm all listed leaders have a geographic location	No	
8.5.2.2. Confirm all listed leaders have a netID / callsign	No	

Table 6. Goals Nine and Ten Hierarchy Lists.

Goal	Implemented?	Source File
G9. Save / export operations order	Yes	D_OrderData.java
9.1. Save operations data to file	Yes	
9.1.1. Generate XML / X-BML file	No	
9.1.1.1. Set file password	No	
9.1.2. Write data to file by data type	Yes	
9.1.3. Encrypt file	No	
9.2. Automatically save operations order	Yes	T_AutoSave.java
9.2.1. Check to see previous saved time	Yes	
9.2.2. Save order as autosave.order	Yes	
9.3. Export operations order as human-readable HTML	Partial	L_HTMLGenerator.java
9.3.1. Export all data elements by readable-text property	Partial	
9.3.2. Organize human-readable elements into 5 Paragraph Format	Partial	
9.3.2.1. Display 5 Paragraph Format in TOOTH	Partial	V_HTMLView.java
9.4. Export operations order as readable Word document	No	
9.5. Export map data as overlay file	No	
G10. Manage device power consumption	Yes	T_PowerMgmt.java
10.1. Check battery status	Yes	
10.2. Prompt user for low-power mode	Yes	
10.2.1. Decrease screen brightness	Yes	
10.2.2. Autosave order	Yes	
10.2.3. Prompt to turn off GPS	Partial	

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. DATA RELATIONSHIP GRAPH



THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Adkison, Jesse D. 2015. “Data Supporting Mobile Application Development for Use within the Marine Air-Ground Task Force.” Master’s thesis, Naval Postgraduate School.

Anderson, Brett. 2010. “Benefits of Using a Gantt Chart.” hubPages. https://hubpages.com/business/Benefits-of-using-a-Gantt-Chart.

The Basic School. 2015a. “Combat Orders Foundations.” In *Basic Officer Course Student Handbook*. B2B2377. Quantico, VA: United States Marine Corps.

———. 2015b. “Officership Foundations.” In *Basic Officer Course Student Handbook*. B1X0099XQ. Quantico, VA: United States Marine Corps.

Börner, Katy, and David E. Polley. 2014. *Visual Insights : A Practical Guide to Making Sense of Data*. Cambridge, MA: MIT Press. http://cns.iu.edu/docs/presentations/2013-borner-visualinsights-cs10k.pdf.

Braude, Eric J. 2001. *Software Engineering: An Object-Oriented Perspective*. New York, NY: John Wiley & Sons.

Buczkowski, Aleks. 2015. “Why Would You Use OpenStreetMap If There Is Google Maps?” Goawesomeness.com. http://geoawesomeness.com/why-would-you-use-openstreetmap-if-there-is-google-maps/.

Cipeluch, Blazej, Ricky Jacob, Adam Winstanley, Peter Mooney, Blazej Ciepluch, Ricky Jacob, Adam Winstanley, and Peter Mooney. 2010. “Comparison of the Accuracy of OpenStreetMap for Ireland with Google Maps and Bing Maps.” *Ninth International Symposium on Spatial Accuracy Assessment in Natural Resuorces and Enviromental Sciences*, 337. http://eprints.nuim.ie/2476/.

Clausewitz, Carl von. 1989. *On War. Ed. and Transl. Michael Eliot Howard and Peter Paret*. Princeton, NJ: Princeton University Press.

Comptroller. 2016. *FY 2017 Department of Defense Military Personnel Composite Standard Pay and Reimbursement Rates*. Washington, D.C.: Department of Defense.

Cormen, Thomas H, Charles E Leiserson, and Ronald L Rivest. 2001. *Introduction to Algorithms.* 2nd ed. Boston, MA: MIT Press.

Creveld, Martin L Van. 1985. *Command in War*. Boston, MA: Harvard University Press.

Department of the Army. 1997. *Staff Organization and Operations.* FM 101-5. Washington, D.C: Department of the Army.

———. 2004. *Operational Terms and Graphics.* FM 1-02. Washington, D.C: Department of the Army.

———. 2014. *Insurgencies and Countering Insurgencies.* FM 3-24. Washington, D.C: Department of the Army.

Expeditionary Warfare Collaborative Team. 2012. *Bold Alligator 2012 Final Report*. Washington, D.C.: Expeditionary Warfare Collaborative Team.

Ferrucci, David, Anthony Levas, Sugato Bagchi, David Gondek, and Erik T. Mueller. 2013. "Watson: Beyond Jeopardy!" *Artificial Intelligence* 199–200. Elsevier B.V.: 93–105.

Filiberti, E. J. 1987. "The Standard Operations Order Format: Is Its Current Form and Content Sufficient for Command and Control?" Master's thesis, United States Army Command and General Staff College.

Franklin, Jude E., Cora Lackey Carmody, Karl Keller, Tod S. Levitt, and Brandon L. Buteau. 1988. "Expert System Technology for the Military: Selected Samples." *Proceedings of the IEEE* 76 (10): 1327–66.

GlobalSecurity.org. 2017. "Marine Corps Special Operations Command Table of Organization." http://www.globalsecurity.org/military/agency/usmc/images/mcsocom-det-org.jpg.

Google. 2017. "Android Developers API Guide." https://developer.android.com/guide/index.html.

Goolsbee, Austin. 2002. "The TurboTax Revolution? Evaluating the Ability of Technology to Solve the Tax Complexity Dilemma." Chicago, IL.

Grossman, Dave, and Loren W Christensen. 2007. *On Combat: The Psychology and Physiology of Deadly Conflict in War and in Peace*. Belleville, IL: PPCT Research Publications.

Hans, Robert T. 2013. "Work Breakdown Structure : A Tool For Software Project Scope Verification." *International Journal of Software Engineering & Applications (IJSEA)* 4, no. 4: 19–25.

Hartmann, Timo, Hendrik Van Meerveld, Niels Vossebeld, and Arjen Adriaanse. 2012. "Aligning Building Information Model Tools and Construction Management Methods." *Automation in Construction* 22. Elsevier B.V.: 605–13.

Highsmith, Jim. 2004. *Agile Project Management: Creating Innovative Products*. Boston, MA: Addison-Wesley.

Kewley, Robert H., and Mark J. Embrechts. 2002. “Computational Military Tactical Planning System.” *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* 32, no. 2: 161–71.

Kiser, Matt. 2016. “Introduction to Natural Language Processing (NLP).” Algorithmia. http://blog.algorithmia.com/introduction-natural-language-processing-nlp/.

Lopes, C. Todd. 2015. “TRADOC Opens App Store.” Army.mil. https://www.army.mil/article/157984/TRADOC_opens_app_store.

Marine Air Ground Task Force Training Center. 2017. “BSC MTWS.” Marines.mil. http://www.29palms.marines.mil/Staff/G3-Operations-and-Training/MAGTFTC-Simulations/Staff-Training/MTWS/.

Miller, Christopher C. 2006. “A Beast in the Field: The Google Maps Mashup as GIS/2.” *Cartographica: The International Journal for Geographic Information and Geovisualization* 41, no. 3: 187–99.

Mohn, Howard L. 1994. “Implementation of a Tactical Mission Planner for Command and Control of Computer Generated Forces in MODSAF.” Master’s thesis, Naval Postgraduate School.

Morris, J. 2011. *Android User Interface Development Beginner’s Guide*. Birmingham, UK: Packt Publishing.

NASA. 2017. “NASA World Wind.” https://worldwind.arc.nasa.gov/.

NATO Military Agency for Standardization. 2000. “Formats for Orders and Designation of Timings, Locations and Boundaries (Stanag 2014).” Brussels, Belgium: NATO Military Agency for Standardization.

NATO Research and Technology Organization. 2012. *Coalition Battle Management Language (C-BML)*. Hanover, MD: NASA Center for AeroSpace Information.

NAVAIR. 2015. “Strike Planning and Execution Systems.” http://www.navair.navy.mil/index.cfm?fuseaction=home.display&key=D0B91B0C-3FA3-4ECA-BADE-CA8F7C3A9825.

———. 2016. “KILSWITCH Application Program Interface.” Data files. China Lake, CA: NAVAIR.

NGA. 2016. “FalconView.” https://www.nga.mil/ProductsServices/Pages/-FalconView.aspx.

Pullen, J. Mark, Ababneh, M., Singapogu, S., Brown, R., Murphy, B., Hall P., and Ave, H.. 2011. “Testing a NATO OPORD Schema with C-BML.” George Mason University, Fairfax, VA. http://netlab.gmu.edu/pubs/11E-SIW-014.pdf.

Pullen, J., Hieb, M., and Levine, S. 2007. “Joint Battle Management Language (JBML)-US Contribution to the C-BML PDG and NATO MSG-048 TA.” *IEEE European Simulation Interoperability Workshop*. http://netlab.gmu.edu/pubs/07E-SIW-029.pdf.

Serbest, Fikret. 1994. “An Automated Tactical Operations Command, Control, Communications, and Intelligence Planning Tool Using Hyper-NPSNET.” Master’s thesis, Naval Postgraduate School.

Smith, Major Matthew L. 1989. “The Five Paragraph Field Order: Can a Better Format Be Found to Transmit Combat Information to Small Tactical Units?” Master’s thesis, United States Army Command and General Staff College.

Starcke, K., and Brand, M. 2012. “Decision Making under Stress: A Selective Review.” *Neuroscience and Biobehavioral Reviews* 36 (4). Elsevier Ltd: 1228–48. doi:10.1016/j.neubiorev.2012.02.003.

tandef. 2012. “Lessons Learned Command Post Exercise Garuda Shield.” Blog post. https://tandef.wordpress.com/2012/07/31/lesson-learn-pelatihan-interoperability-dan-interagency-melalui-command-post-exercise-garuda-shield/.

Tausworthe, Robert C. 1979. “The Work Breakdown Structure in Software Project Management.” *Journal of Systems and Software* 1: 181–86. doi:10.1016/0164-1212(79)90018-9.

U.S. Joint Forces Command. 2011. *Joint Operation Planning*. JP 5-0. Washington, D.C: U.S. Joint Forces Command.

United States Marine Corps. 2017. “USMC Unit Directory.” Marines.mil. Accessed March 3. http://www.marines.mil/Units/srtype/Infantry/.

———. 2010. *MCWP 5–1: Marine Corps Planning Process*. Quantico, VA: United States Marine Corps.

———. 2015. *Marine Corps Concept for Command and Control*. Quantico, VA: United States Marine Corps.

Weaver, P. 2012. “The Origins of Modern Project Management.” *Science*, 1401(2003), 557–575. https://doi.org/10.1126/science.1089370.

Wu, C. Thomas. 2006. *An Introduction to Object-Oriented Programming with Java*. 4th ed. New York, NY: McGraw-Hill.

Zsambok, Caroline E, and Gary Klein. 2014. *Naturalistic Decision Making*. New York: Psychology Press.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California